

**FAKULTI SAINS KOMPUTER DAN TEKNOLOGI MAKLUMAT**

**API Management Platform**

COURSE : Advanced Software Engineering

CODE : SIM5202 - 1

LECTURER : Dr. Ng Keng Yap

**STUDENT NAME MATRIC NO.**

SYAFIQ ALI BIN OTHMAN GS53930

IZUKA JOSEPH IHEANACHO GS51769

CHINEDU STEPHEN NWISU GS54880

GARBA MUHAMMAD GS54347

LIM SWEE PHANG GS54609

Table of Contents

[Table of Contents 1](#_Toc9767021)

[List of Figures & Tables 2](#_Toc9767022)

[Chapter 1: Introduction 3](#_Toc9767023)

[1.1 Background 3](#_Toc9767024)

[1.2 Problem Statement 3](#_Toc9767025)

[1.3 Project Objective 4](#_Toc9767026)

[Chapter 2: Literature Review 5](#_Toc9767027)

[2.1 Application Programming Interface (API) 5](#_Toc9767028)

[2.2 Application Programming Interface(API) documentation 6](#_Toc9767029)

[Chapter 3: Methodology 8](#_Toc9767030)

[3.1 Project Approach 8](#_Toc9767031)

[3.2 Project Tools and Platform 8](#_Toc9767032)

[3.2.1 Git – Version control 8](#_Toc9767033)

[3.2.2 Github – Online repositories 9](#_Toc9767034)

[3.2.3 GitKraken – Kanban board integrate with Github 9](#_Toc9767035)

[3.2.4 Heroku – Cloud platform for build and deliver apps 9](#_Toc9767036)

[3.2.5 TravisCI – Test and deploy code 9](#_Toc9767037)

[3.3 Project Method 10](#_Toc9767038)

[Chapter 4: Results and Evaluation 12](#_Toc9767039)

[Chapter 5: Conclusion 16](#_Toc9767040)

[REFERENCE: i](#_Toc9767041)

[APPENDIX ii](#_Toc9767042)

# 

# List of Figures & Tables

[Figure 2.1 Privacy-preserving models for fog-based IoT applications 5](#_Toc514406018)

[Table 2.1 Summary of Related Works. 6](#_Toc514406018)

Figure 3.1 Illustration of Three-Layer storage framework based on fog computing 9

Figure 3.2 Diagram of stored procedure 11

Figure 3.3 Diagram of download procedure 12

Figure 3.4 Hash-Solomon matrix operation 13

Figure 3.5 Relationship of a number of data blocks (k), redundant data blocks (m), and storage ratio (r). 15

Figure 3.6 Diagram of the inﬂuence of the number of data blocks (k) to the efﬁciency of storage and coding 15

Figure 3.7 Inﬂuence of the number of data blocks (k) to the efﬁciency of storage and coding 16

Figure 4.1 Relationship between the time of decoding and the number of k 18

Figure 4.2 The local storage volume of different files 18

Figure 4.3 Relationship between the time of encoding and the number of k 19

Figure 4.4 Relationship between time of decoding and the number of removed data 19

Figure 4.5 Cauchy matrix vs Vandermonde matrix 20

Figure 4.6 Relationship between time of encoding and word size of Galois field. 20

# Chapter 1: Introduction

## Background

As the rapid advancement of cloud computing, more and more web services offer application interfaces called web APIs for application developers to use. Many famous web service providers such as Facebook, Yahoo, and Google provide a wide range of web APIs.

With the increasing use of distributed web APIs, the access control of web APIs becomes more and more critical and has different process flows based on different scenarios. In most conditions, the web API access control focuses on the user level. The third-party application redirects resource owner's user-agent to the server's resource owner authorization endpoint to request her to sign in using her username and password and if successful, asks her to approve a third-party application to access her personal resources. The usage of these application interfaces is basically free, and the scenario is mainly limited to popular enterprise web services.

It is believed that API is becoming one of the most powerful tools that services, components, and devices have ever had. Developers today can create new applications on top of them to unleash a great potential and contribute in creating an ecosystem around a service.

## 1.2 Problem Statement

Nowadays, organizations are implementing microservices architecture in order to meet rapid changes in customer demands. HTTP-based APIs is preferred method for synchronous interaction between microservices architecture. These APIs are the glue that sticks all of the microservices together. Organization needs strategies to manage their APIs and also make sure their API is used in compliance with corporate policies and allows governance by suitable levels of security. In order to manage these APIs, an organization needs a platform.

## 1.3 Project Objective

The main objective of this project is:

* To provide a platform for an organization to manage their APIs and enables for other modules/subsystem to integrate.

# Chapter 2: Literature Review

## 2.1 Application Programming Interface (API)

(Souza, David, & John, 2004) An API is a well-defined interface supported by the underlying programming language, that allows one software component to access programmatically another component. A more “formal” definition is provided by the Software Engineering Institute is the following (De Souza, Redmiles, Cheng, Millen, & Patterson, 2004):

Application Programming Interface (API) is an older technology that facilitates exchanging messages or data between two or more different software applications. API is the virtual interface between two interworking software functions, such as a word processor and a spreadsheet. An API is the software that is used to support system-level integration of multiple commercial-off-the-shelf (COTS) software products or newly developed software into existing or new applications.

The field of software engineering observed some technical problems caused by interdependencies and developed tools, approaches, and principles to deal with them. Configuration management and issue-tracking systems are examples of such tools, and the adoption of software development processes (De Souza, Redmiles, & Dourish, 2003; De Souza, Redmiles, Mark, Penix, & Sierhuis, 2003)

(Parnas, 2002) proposed the idea of information hiding as one of the most important and influential principles used to manage dependencies. By this principle, software modules should be both “open (for extension and adaptation) and closed (to avoid modifications that affect clients)” (Larman, 2001). Information hiding aims to provide a principle that guides the decomposition of a software system into pieces (called modules) that decreases the dependency (or coupling) between any two modules. Changes to one module do not severely impact other modules by this principle. This principle motivates several mechanisms in programming languages that provide flexibility and protection from changes, including data encapsulation, interfaces, and polymorphism (Larman, 2001). (Fowler, 2002) emphasized that separating interface specifications from their implementation is a growing trend in software design. Additionally, interface specifications are also helpful in the coordination of developers working with different components:

(Grinter, Herbsleb, & Perry, 1999) *expressed that*

*Interface specifications play the well-known role of*

*helping to coordinate the work between developers of*

*different components. If the designers of two*

*components agree on the interface, then design of the*

*internals of each component can go forward relatively*

*independently. Designers of component A need not*

*know much about the design decisions made about*

*component B, so long as both sides honor their well-*

*specified commitments about how the two will hook*

*together.*

(Souza et al., 2004) identified two main functions performed by APIs in the software development process regarding MCW and its hosting organization: they function as contracts among stakeholders and, in addition, as reifications of the organizational boundaries. APIs concurrently allow and constrain collaboration among software developers, contradicting the common wisdom among software developers that APIS are only beneficial.

Furthermore,(Parnas, 2002) stated that APIs facilitate collaboration during the process of breaking a system into units that can be developed independently because (i) as reifications, they enforce the organizational boundaries of team membership, and (ii) as contracts they establish a shared understanding of what needs to be done and at some level formalize this agreement.

## 2.2 Application Programming Interface(API) documentation

Application programming interfaces (APIs) expose services or data provided by a software application through a set of predeﬁned resources, such as methods,objects, or URIs (Stylos, Faulring, Yang, & Myers, 2009). By using these resources, other applications can access the data or services without having to implement the underlying objects and procedures. APIs are central to many modern software architectures, as they provide high-level abstractions that facilitate programming tasks, support the design of distributed and modular software applications and the reuse of code (Robillard, 2009).

As Myers and Stylos (2016) point out, all modern software makes heavy use of APIs. Instead of programming functionality from scratch, the fundamental task of software developers now often is to ‘‘stitch together’’ functionality that existing APIs provide (Stylos, 2009, p. 4). The question of API learnability is closely linked to the more general issue of API usability (Clarke, 2004; McLellan, Roesler, Tempest, & Spinuzzi, 1998). Some APIs are more diﬃcult to learn and to use than others (Myers & Stylos, 2016). While some of this diﬃculty is likely due to properties inherent to the API, such as the domain it covers, speciﬁc aspects of API design contribute to the diﬃculty as well. By now, many studies have identiﬁed and investigated relevant design aspects and demonstrated that APIs can be learned and put to use more easily if the API design is tuned to the way developers work and matches the expectations they form toward an API (McLellan et al., 1998; Stylos & Clarke, 2007; Zibran, 2008). Besides aspects of API design, the growing interest in API usability has also generated growing interest in API documentation which reﬂects the fact that documentation has critical impact on usability by enabling users to solve taskseﬀectively and eﬃciently (Alexander, 2013; Guillemette, 1989; Redish, 2010). The ﬁndings from the literature suggest that API documentation plays an important role in learning and using APIs (Dagenais & Robillard, 2010;Lethbridge, Singer, & Forward, 2003), but often does not seem to ﬁt the information needs and expectations of API users.

It has also been observed that developers tend to prefer information sources outside the oﬃcial API documentation in case they face a problem that they need to solve. For example, Parnin (2013) reports results from an informal study that tracked the search behavior of Android developers over a period of 11 weeks. The study found that developers visited StackOverﬂow, one of the currently most active developer forums, three times more often than the Google site hosting the oﬃcial Android documentation. Parnin’s results converge with an observation emphasized by Treude, Barzilay, and Storey (2011) according to which answers on StackOverﬂow often become a substitute for oﬃcial product documentation. All these ﬁndings suggest that more research is necessary to better understand which information is necessary to enable developers to get into a new API eﬃciently and eﬀectively. Researching these information needs is a crucial part of audience analysis. The work by (Robillard, 2009; Robillard & DeLine,2011; Uddin & Robillard, 2015) on documentation related obstacles and deﬁciencies with existing API documentation that hinder learning extends those studies as well as other previous studies on API documentation in taking a broader approach that discusses information needs and information-seeking behavior of developers in the context of the process of learning an API.

Learning an API involves understanding the functionality exposed by an API as well as learning the API elements and how to coordinate them in order to bring the required functionality about (Stylos & Myers, 2006). Being an everyday activity of developers, such learning can be conceived of as ‘‘self-directed learning’’ in the sense of Knowles (1975). According to Knowles(1975), self-directed learning is a process in which individuals take the initiative in, among other things, formulating learning goals, identifying useful resources for learning, and adopting appropriate learning strategies (p. 18). Our study helps to understand information needs of software developers by eliciting data on learning goals developers set for themselves when starting the learning pro-cess, as well as speciﬁc activities they initiate and information resources they turnto in order to accomplish those goals.

# Chapter 3: Methodology

This chapter focuses on describing and discusses the method used for the project.

## 3.1 Project Approach

This project involved building a website to provide a platform for an organization to manage their APIs. The development process includes the Scrum framework for managing web development and using DevOps for continuous integration and continuous delivery of the system. There are several tools and platform used for the project to accelerate the development process. The approach used for developing the website platform is COTS product reuse which the system developed by configuring and integrating existing application system. The project also used application framework like Laravel, which is the Model View Controller framework to create demo application to test the API management platform.

## 3.2 Project Tools and Platform

List of tools and platforms used for the project and their functions are described below:

1. Git – Version control
2. Github – Online repositories
3. GitKraken – Kanban board integrate with Github
4. Heroku – Cloud platform for build and deliver apps
5. TravisCI – Test and deploy code
6. Digital Ocean droplet – Cloud platform for server

## 3.2.1 Git – Version control

Git is a freely available open source distributed version-control system designed to handle everything, especially tracking changes in files swiftly and efficiently. It is designed with the mind to coordinating and provide collobaration in work among programmers. Particularly, we use git to clone our project’s Github repository to Digital Ocean droplet for deployment.

## 3.2.2 Github – Online repositories

In this project, we decided to use Github for our teammates for collaboration. Github is web-based version control using Git as underlying technology. It offers all features provided by Git but also extends to other features such as issue tracker, project board, feature requests, 3rd party apps/service integration, dependencies security vulnerabilities checker, webhooks, and wikis for every project.

## 3.2.3 GitKraken – Kanban board integrate with Github

GitKraken is a freemium, fully-featured, cross-platform Git desktop GUI client by Axosoft. It makes git commands and processes easy, fast, and intuitive by providing visually attractive interface for branching, merging and your commit history, seeing branches owners with profile icons, and even management of pull requests all locally, without the use of browser.

It is built with nodegit so you don't even need to install Git on your system.

Other than that, it also has built-in merge tool and provide easy keyboard shortcuts.

## 3.2.4 Heroku – Cloud platform for build and deliver apps

Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. Heroku Pipelines allows multiples apps to have the same codebase for review, development, staging, and production environments to support, manage and visualize continuous delivery. Promoting tested code from one stage to the next can be done manually or automatically and is nearly instantaneous. For Heroku apps connected to GitHub, Heroku can manually or automatically spin up a temporary test app on a unique URL for every opened pull request (PR). The temporary app is auto-updated on every commit, so instead of guessing about what the code might do, reviewers can actually try the changes in a browser. We used Heroku to assist in CI/CD process.

## 3.2.5 TravisCI – Test and deploy code

Travis CI is a continuous integration online service that can be used to build and test software projects hosted at GitHub. User need to add a file named .travis.yml, to the root directory of the repository in order to start using it. It specifies the programming language used, the desired building and testing environment (including all the dependencies). We use Travis CI along with Heroku to make the whole CI/CD process streamlined. When we commit any changes to GitHub, it will be triggered to run the automated test and building process by utilizing webhooks technology. When the process has completed, developer(s) will be notified whether it is success or failure through preconfigured ways such as emails. That’s how we know if someone broke the code(build failed).

## 3.3 Project Method

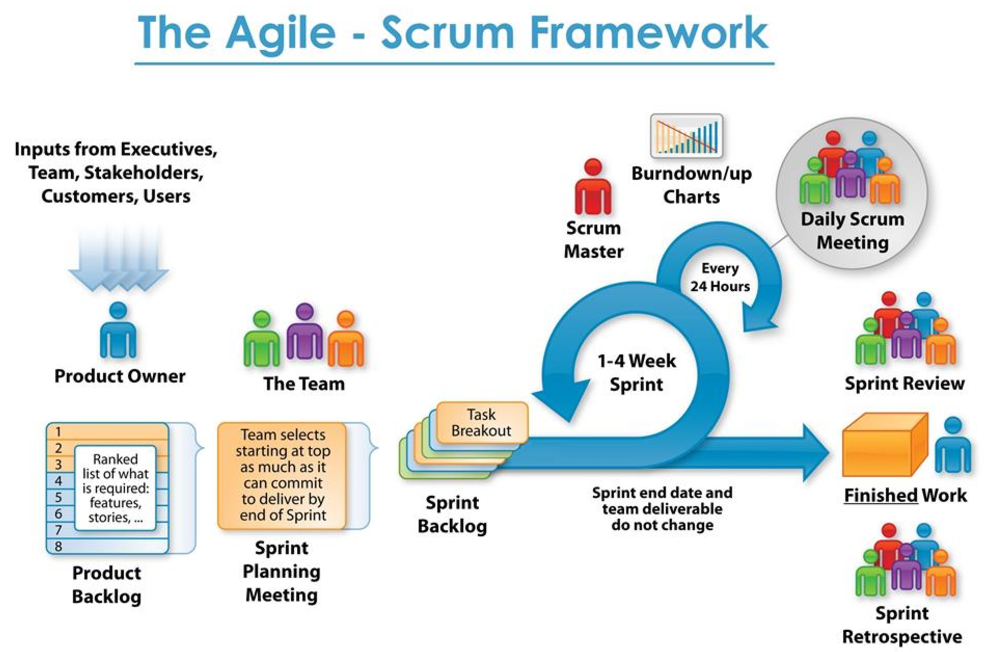


Figure 3.1: Scrum Framework (Nuno Oliveira, 2017)

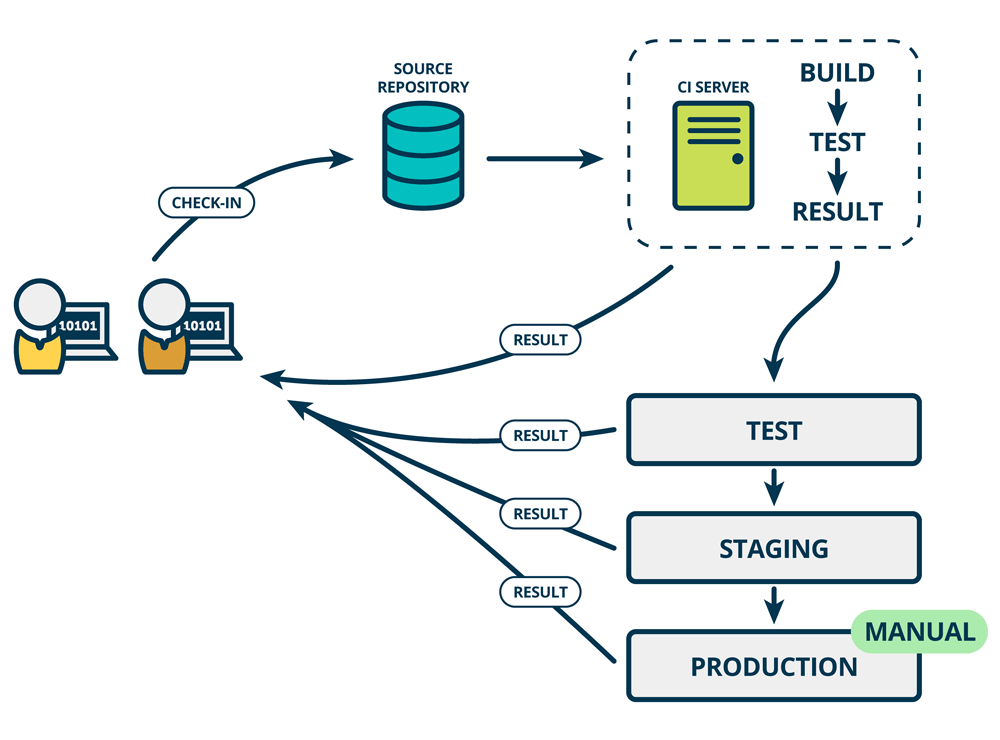


Figure 3.2: DevOps operation (Suzie Prince, 2016)

1. The Product Owner creates a list of requirements. The team member and scrum master write the user stories and put it in Product Backlog. GitKraken is being used to record the product backlog.
2. Next, we performed sprint planning meeting to decide on the contents of the next sprint. Took several item from Product Backlog, add to the Sprint Backlog for the next sprint.
3. We performed DevOps operation during sprint. Used tools and platform during the sprint to accelerate the process.
4. Performed daily scrum meeting to answer the following three question:
   * What did I do yesterday that helped the Development Team meet the Sprint Goal?
   * What will I do today to help the Development Team meet the Sprint Goal?
   * Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?
5. At the end of each Sprint there is “review meeting” and retrospective.

# Chapter 4: Results and Evaluation

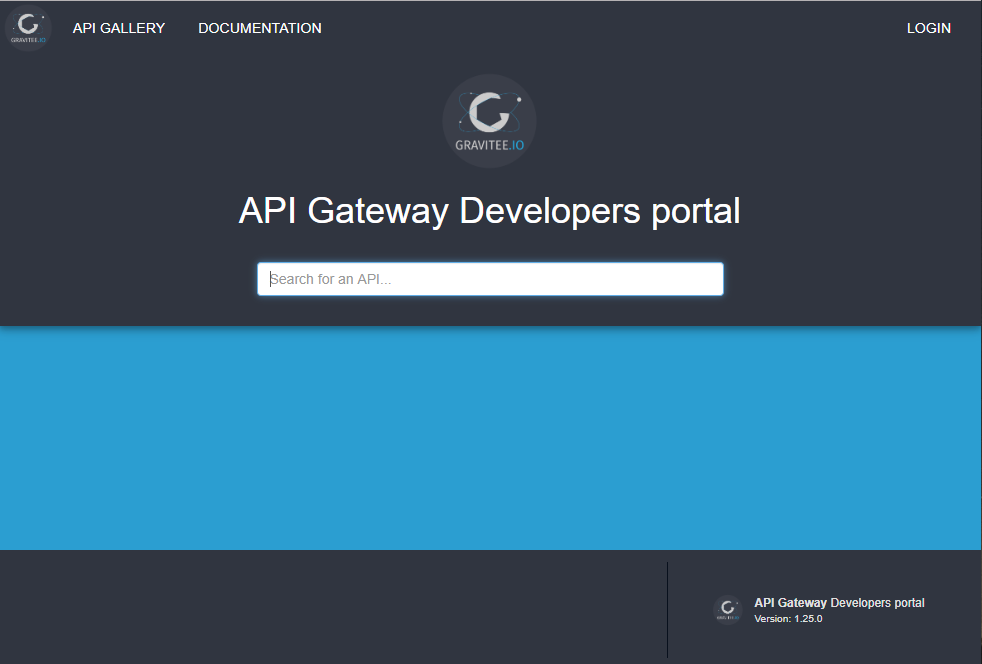


Figure 4.1: API Management Platform. (Portal)

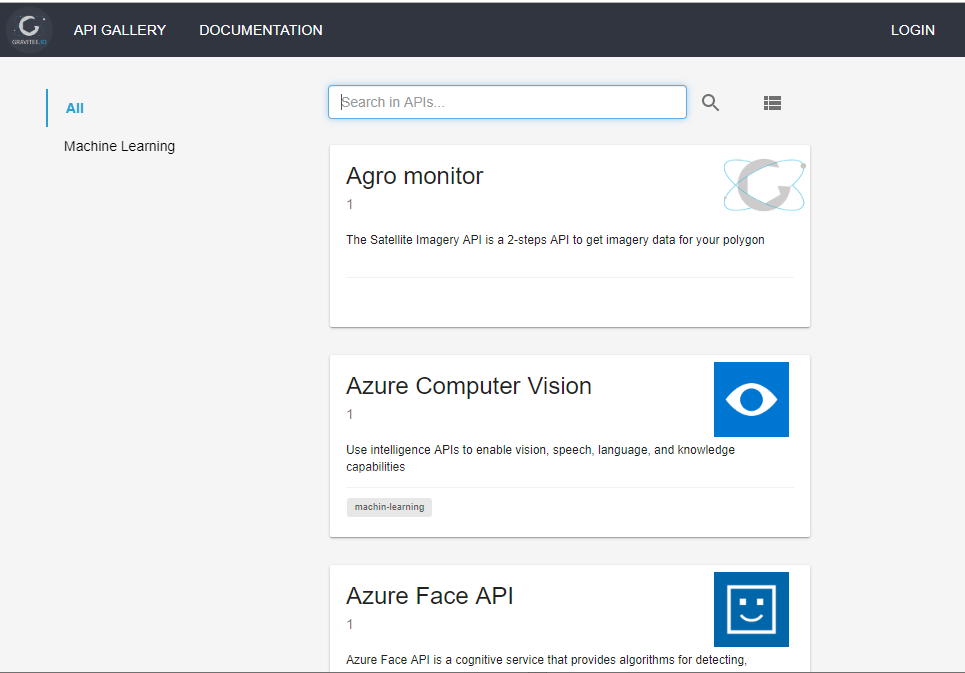


Figure 4.2: API Management Platform. (API Gallery)

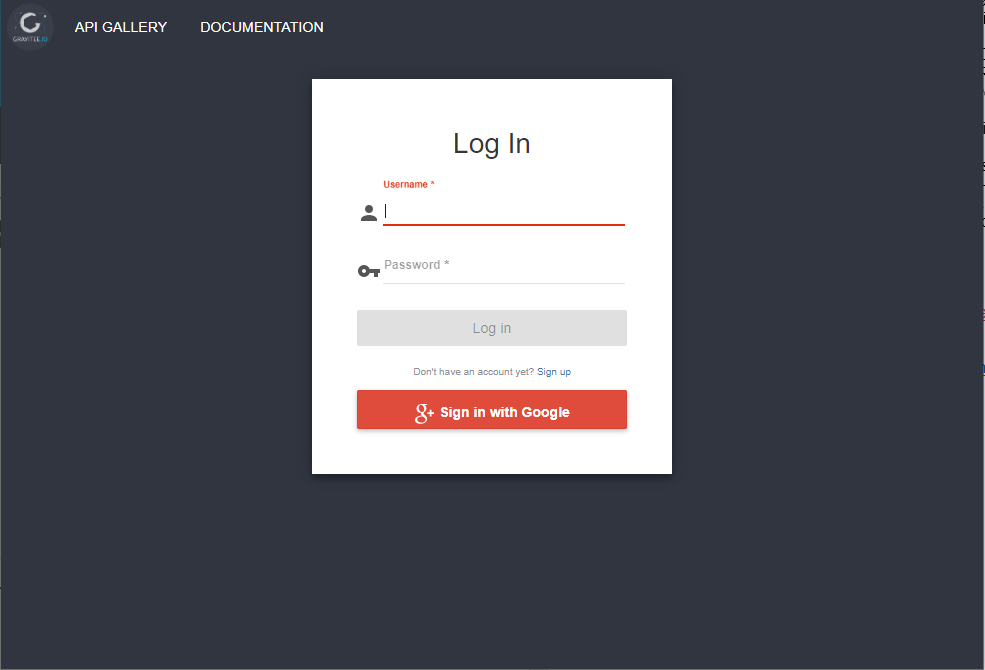


Figure 4.2: API Management Platform. (Login prompt)

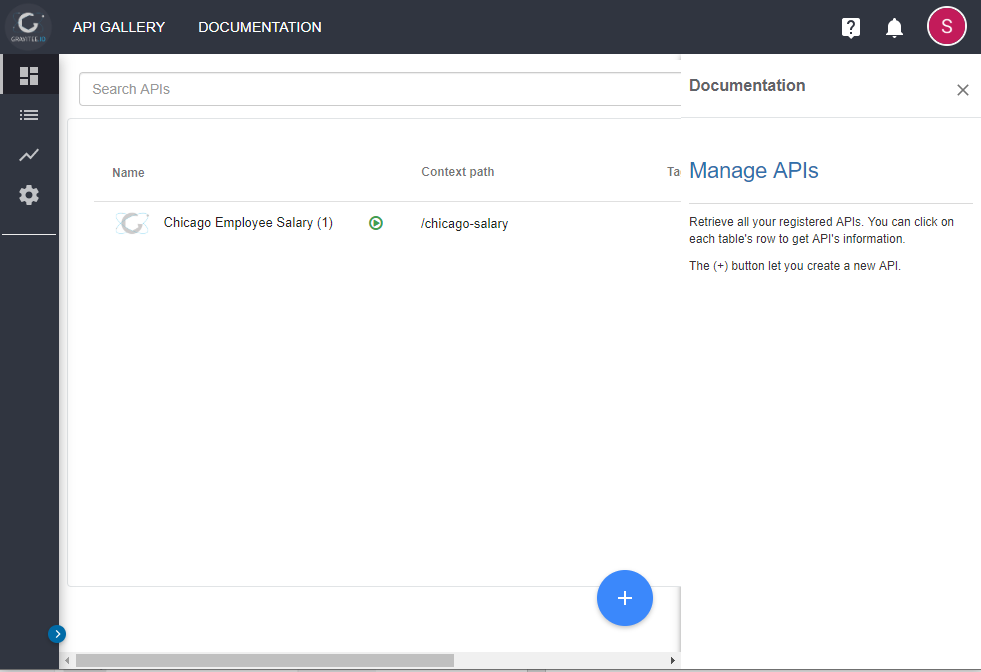


Figure 4.3: API Management Platform. (Manage API)

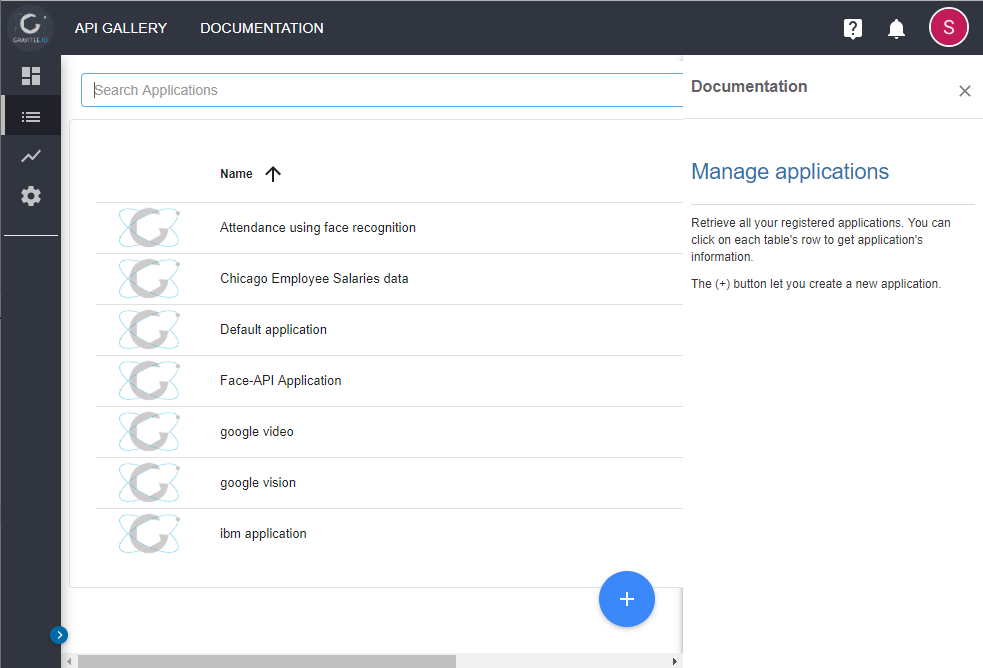


Figure 4.4: API Management Platform. (Manage Application)

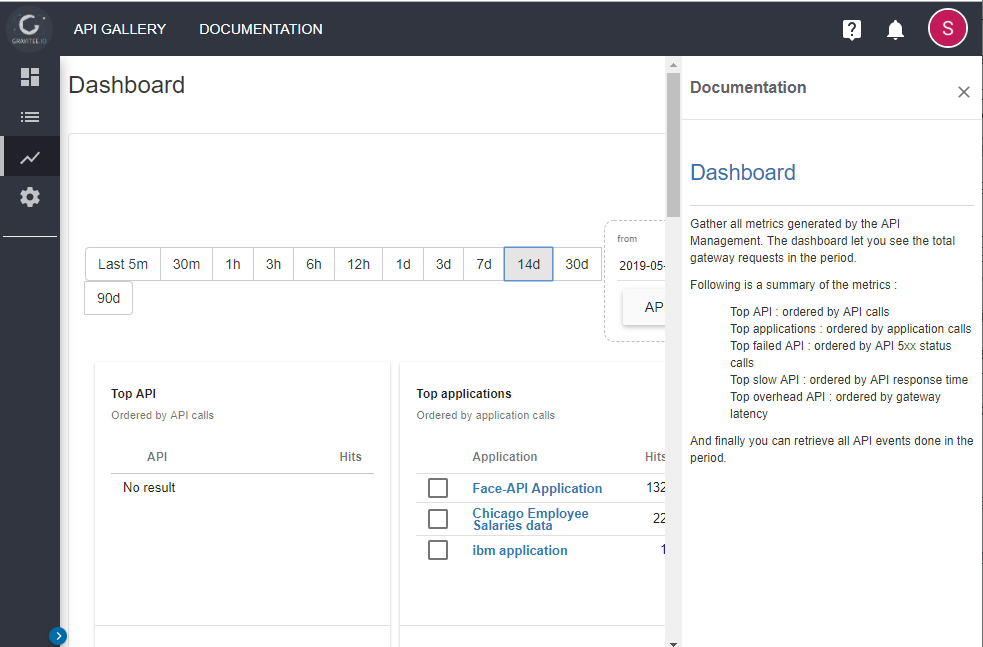


Figure 4.5: API Management Platform. (Dashboard)

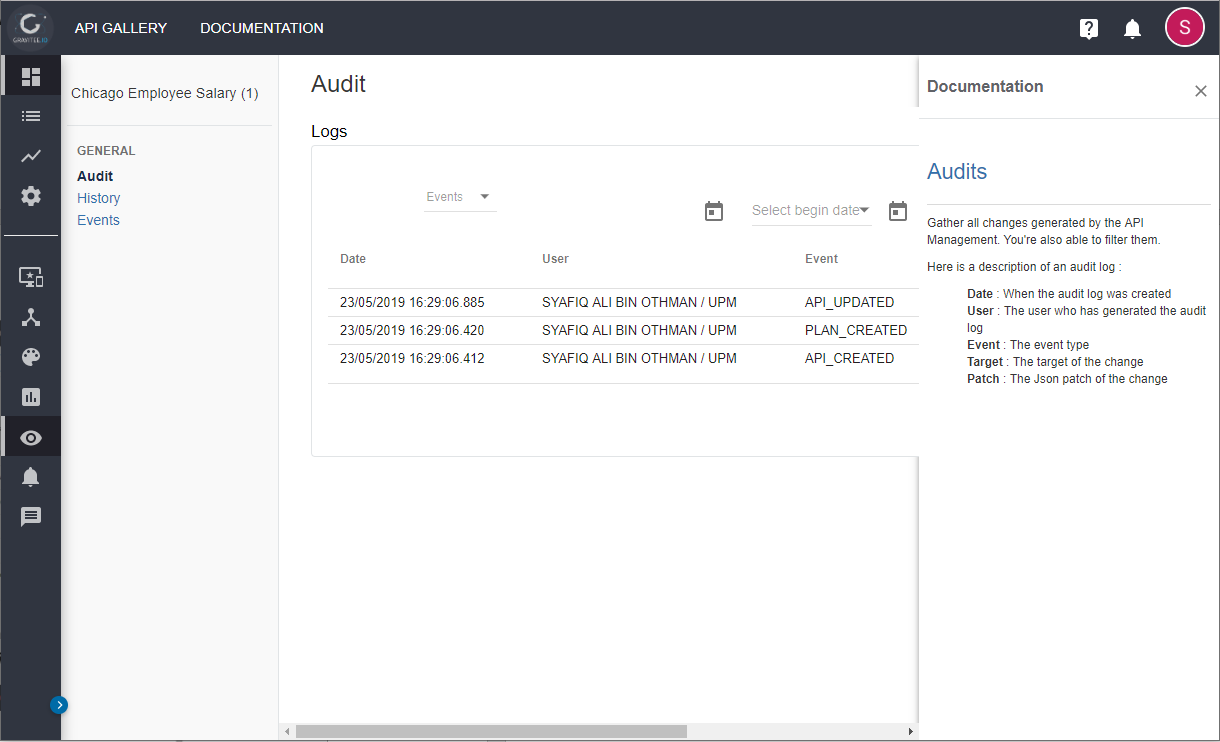


Figure 4.6: API Management Portal. (Audit Trail)

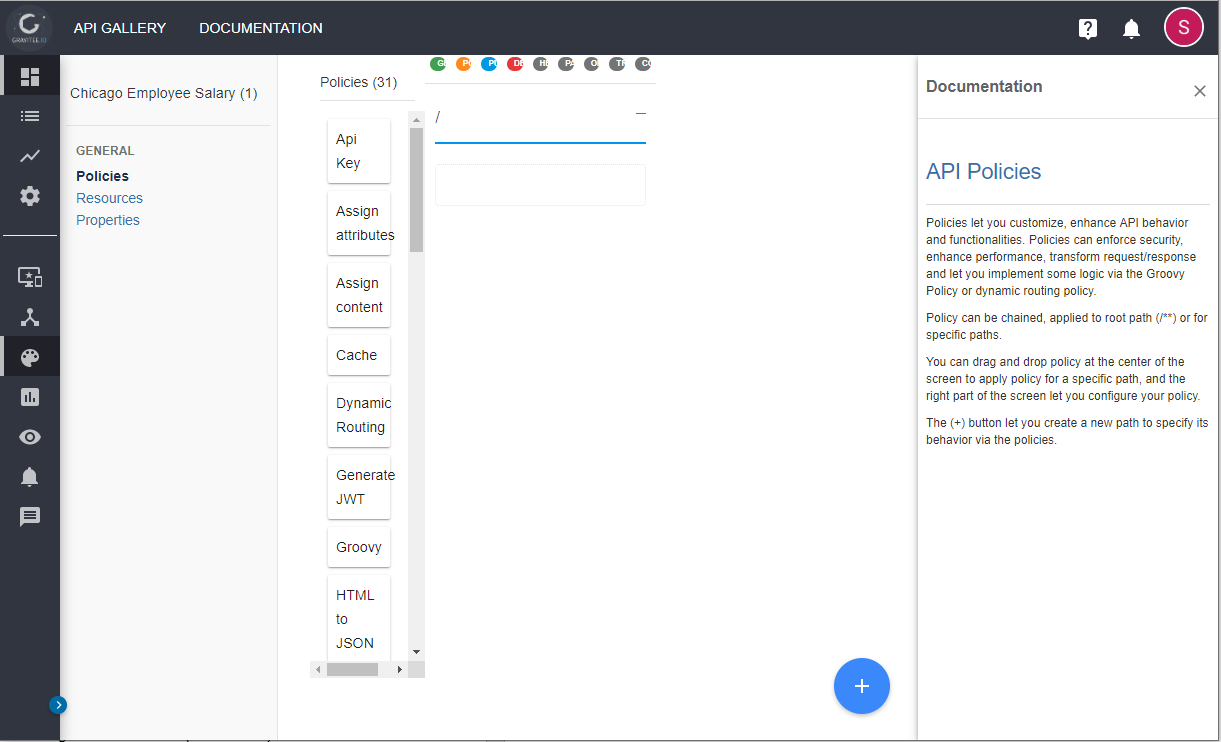


Figure 4.7: API Management Portal. (API Policies)

# Chapter 5: Conclusion

# REFERENCE:

De Souza, C. R. B., Redmiles, D., Cheng, L.-T., Millen, D., & Patterson, J. (2004). How a good software practice thwarts collaboration: the multiple roles of APIs in software development. *ACM SIGSOFT Software Engineering Notes*, *29*(6), 221–230.

De Souza, C. R. B., Redmiles, D., & Dourish, P. (2003). Breaking the code, moving between private and public work in collaborative software development. In *Proceedings of the 2003 International ACM SIGGROUP conference on Supporting group work* (pp. 105–114).

De Souza, C. R. B., Redmiles, D., Mark, G., Penix, J., & Sierhuis, M. (2003). Management of interdependencies in collaborative software development. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* (pp. 294–303).

Fowler, M. (2002). Public versus published interfaces. *IEEE Software*, *19*(2), 18–19.

Grinter, R. E., Herbsleb, J. D., & Perry, D. E. (1999). The geography of coordination: Dealing with distance in R&D work. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work* (pp. 306–315).

Larman, C. (2001). Protected variation: The importance of being closed. *IEEE Software*, *18*(3), 89–91.

Parnas, D. L. (2002). On the criteria to be used in decomposing systems into modules. In *Software pioneers* (pp. 411–427). Springer.

Souza, C. R. B. De, David, L. C., & John, M. (2004). Sometimes You Need to See Through Walls — A Field Study of Application Programming Interfaces, *6*(3), 63–71.

<https://itnext.io/scrum-and-outsystems-the-perfect-combination-to-bring-value-do-the-business-5a316f2d34a>

<https://www.mindtheproduct.com/2016/02/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/>

# APPENDIX

Documentation for publishing API and consuming API

