FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Participating in the API Economy: An API Lifecycle Analysis

B.Sc. Joan Disho

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Participating in the API Economy: An API Lifecycle Analysis

Partizipation in der API Economy: Eine Analyse des API Lebenszyklus

Author:	B.Sc. Joan Disho
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	M.Sc. Gloria Bondel
Date:	11 June 2018

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 11 June 2018 B.Sc. Joan Disho

Abstract

Application programming interfaces (APIs) are an essential tool, and the market around them is thriving. Opening up APIs has led to a range of new business models, which in turn created a new market with complex relations. This new market is summarized under the term API Economy. The API Economy plays a vital role in the digitalization of today's world since APIs are transformed into digital products, affecting the way organizations cooperate.
In this thesis, I analyze the API lifecycle in the API Economy, which is supposed to help companies to participate in the API Economy network and include examples from companies that already thrive in it.
First, I do systematic literature research to gather information regarding APIs, API Economy, their growth, and trends. Additionally, I analyze the business strategies, monetization models and legal considerations that companies need to use when they plan to have an API. Furthermore, the design techniques and technologies to design, build and deploy an API. Moreover, the marketing and engagement strategies to boost the API traffic in the market and the performance metrics to measure the API growth and success. Last but not least, the reasons and preparations for retirement to keep an excellent reputation in the API community.
Based on this information, I create a thorough lifecycle analysis of the API participating in the API Economy.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays, a company without an application program interface (API), which is an intermediary, that allows applications or software programs to interact with each other, is like the Internet without the World Wide Web. APIs are allowing companies to grow businesses at unprecedented rates by sharing services with external firms. The demands are changing, and every company is under pressure to move quickly toward APIs because competition comes from anywhere, a startup, an established player, a traditional competitor or someone entirely outside the field.

1.2 Problem

I often see two things today when it comes to APIs, either the closed partner-to-partner API model, where a company announces that is sharing data with another one, but not usable unless you are an official partner. Alternatively, the other situation where a company launches a technically great API but does so with no developer business model. There is no clear way, as a guideline, which helps companies participating in the API Economy.

1.3 Goal

By having an API lifecycle analysis in the API Economy, I want to help companies to participate and succeed in the API Economy network.

1.4 Objectives

I aim to answer three research questions (RQs).

1.4.1 RQ1: What is an API and API Economy?

With this research question, I want to to give a short introduction about APIs, their growth in the last 20 years and define the term API Economy.

1.4.2 RQ2: Which are the lifecycle phases of an API participating in the API Economy?

From conception to depreciation, an API part of the API Economy is prone to constant evolution. With this research question, I will show the phases of the API lifecycle and how it appears to be in a top level design.

1.4.3 RQ3: What are the components of each API lifecycle phase?

With this research question, I will do a thorough lifecycle analysis of the API participating in the API Economy and analyzing its components in details in a dedicated chapter.

1.5 Outline

First, I start introducing how I do systematic literature research, to gather information about APIs, API Economy and their growth for the last 20 years. Additionally extracting information that helps me to find out the lifecycle phases of the API which participates in the API Economy, together with the its components. Moreover, in a dedicated chapter analyzing each in details. At the end stating my conclusion and the limitations of this thesis.

Chapter 2

Systematic literature review in API Economy

In this chapter, we will explain how I do systematic literature review (SLR) to find out what are the business strategies, monetization models and legal considerations that companies need to use when they plan to have an API. Moreover, what are the design techniques and technologies to design and build an API? Additionally, what are marketing and engagement strategies to boost the API traffic in the market and the performance metrics to measure the API growth and success? Lastly, what are the reasons for retirement and the preparations to keep an excellent reputation in the API community?
The review will be based on the guidelines provided by Kitchenham. [BK09]

2.1 Systematic Literature Review

SLR stands for Systematic literature review, which is a subclass of literature reviews that collects and rigorously analyzes multiple research results, by identifying and formulating the research questions, critically evaluating the studies that are related to those questions and collecting the relevant information. [1]
The **first step** in conducting a systematic literature review is to create the research questions that will guide the review. The **second step** is to perform a thorough manual search of journals, conferences, in certain digital libraries such as ACM Digital Library, Scopus, IEEE Xplore. The **third step** is about the unbiased inclusion and exclusion criteria in which will allow me to address the research question(s) we are posing. It is important that these inclusion and exclusion criteria are applied consistently throughout the review.

[1]Systematic Review: http://getitglossary.org/term/systematic+review

Lastly, I collect the information from the sources that we selected.[BK09]

2.2 Review Method and Conduct

We will explain each activity in detail and describe how we approach it.

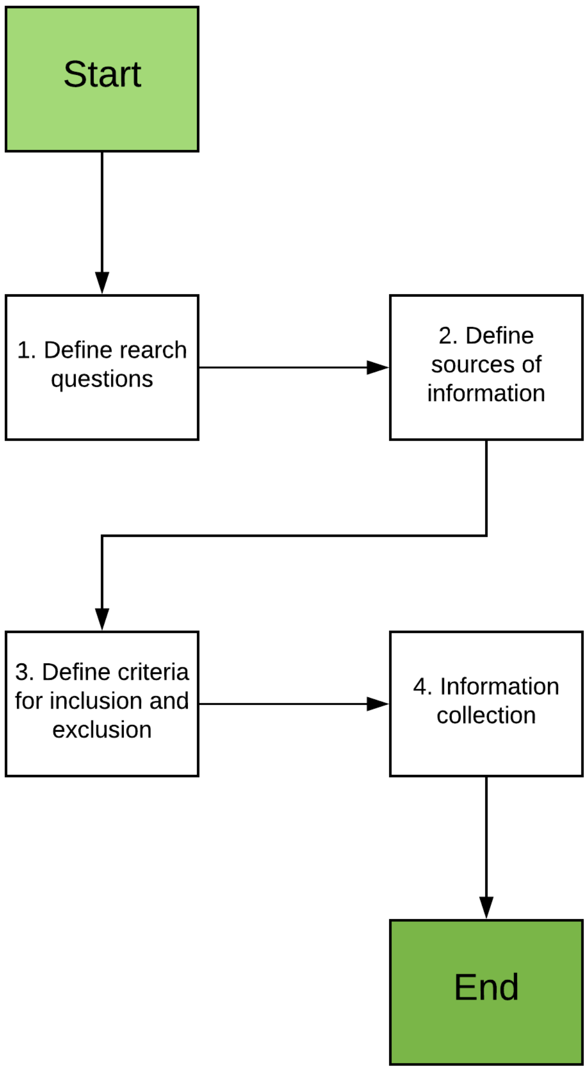

Start
1. Define rearch questions
2. Define sources of information
3. Define criteria for inclusion and exclusion
4. Information collection
End

Figure 2.1: Systematic literature review process

2.2.1 Research questions

Any systematic literature review begins with a series of questions which direct the review. The questions addressed for SRL are as follows:

Q1 What are the aspects that a company should consider when planning to have an API?

Q2 What are the design techniques and technologies to design and build an API?

Q3 What are the marketing and engagement strategies to boost the API traffic in the market?

Q4 What are the metrics to measure the API growth and success?

Q5 What are the retirement reasons and the preparations which can help a company to keep an excellent reputation in the API community?

2.2.2 Search process

The search process will be a manual search for specific conferences, papers, journals, highly cited blog sites and API providers. I include blog sites and API providers because APIs are dynamically changing, new technologies are introduced, new business models are applied, new design implementations are adapted, and those sources can provide the information, faster.
The selected sources where our review will be based is presented in the table 2.1.

ACM Digital Library is chosen because is the world's largest scientific and educational computing society and hosts many scientific papers in software engineering. DBLP, because it has more than 3.66 million journal articles, conference papers, and other publications on computer science. The IEEE Xplore provides access to more than 3.5 million documents from some of the world's most highly cited publications also in computer science and software engineering. Scopus, Google Scholar, is also an important source because they include the results of other major research databases.
ProgrammableWeb, because it is known as the most important registry of the API Economy and also as the world's leading source of news and information about APIs. ProgrammableWeb is also the most widely-cited source of data when it comes to the inclusion of API-related statistics in the mainstream

ID	Sources	Source Type
S1	ACM Digital Library	Digital Library
S2	DBLP Computer Science Bibliography, The	Digital Library
S3	IEEE Xplore / Electronic Library Online (IEL)	Digital Library
S4	Scopus	Digital Library
S5	Google Schoolar	Digital Library
S6	ProgrammableWeb	Conference + Journal
S7	NordicAPI	Conference
S8	IBM API Economy	Blog
S9	API Evangelist	Blog
S10	APIGee	API Provider
S11	RedHat	API Provider
S12	Mulesoft	API Provider

Table 2.1: Sources of information

media, conferences, white-papers, and other forms of research. [2]
Nordic APIs, as the largest community for API practitioners and enthusiasts. Through the global reach of Nordic APIs blog and their conferences held around the world, they help companies make smarter tech decisions using APIs and inspiring better API solutions.
The other sources that are listed in the table are chosen because they are highly cited and are well received by the API community.

[2]About ProgrammableWeb: https://www.programmableweb.com/about

2.2.3 Inclusion and exclusion criteria

The inclusion criteria are as follows:

1. The search results are in English language.
2. The search results are a direct focus in relation to the raised questions.
3. The search results from Digital Databases are the full papers, including the abstract.
4. The search results from conferences and journals are dedicated articles in relation to our research question.

The exclusion criteria are as follows:

1. The search results provide too general information.
2. The search results don't focus specifically on Web API. Our research questions are related only to Web APIs.
3. Duplicated results.

2.2.4 Information collection

After the inclusion/exclusion criteria, I have to define what kind of information are we extracting from our sources. First, I am considering all the articles which are not part of the exclusion criteria. Furthermore, I consider collecting the sources for each article. This will hopefully help me finding useful materials that can better answer our research questions.

Chapter 3

API and API Economy

Before analyzing the lifecycle of APIs participating in the API Economy, I will introduce the concepts of API and API Economy.

3.1 Definition of API

API stands for application programming interface. In a technical sense, API is a machine-readable interface through which software applications make their functionality or data accessible to another authorized application [Kep14]. In other words, API exposes some of the application's internal functions to outside world through a defined interface. In this way, the outside world can use the functionality without knowing how the internals of the application work. The goal of the API is to ease and accelerate the development of software applications by providing a part of its functionality to the outside world, so developers don't lose time implementing the solution themselves. There are different types of APIs, but in this thesis, I will focus on a particular kind of API, the Web API.
An API is a type of interface where the communication takes place on the Internet using specific Web protocols like HTTP. The API defines a set of endpoints that receive requests and send responses in defined rules by SOAP (Simple Object Access Protocol) protocols or REST (Representational State Transfer) or GraphQL (Graph Query Language). The communicated data usually are in XML or JSON format.
An excellent example of an API is the Twitter API [1], where developers can engage with the Twitter platform without any GUI. Twitter gives to developers through its API the possibility to post, retrieve and engage with tweets, access direct messages, upload media content and more. The API has made

[1]Twitter API: https://developer.twitter.com/en/docs.html

possible to develop multiple different Twitter software applications, mobile and desktops like Tweetbot and Twitterrific. The software applications offer different user experience, but they target the same audience as Twitter does, using the same underlying data provided by the Twitter API.
Simplifying it, an API is a “digital glue” that connects different systems and companies while creating new application and partnerships. Now APIs are a big part of the web. In January 2018 there were over 19,000 APIs published by companies for usage in ProgrammableWeb, which is the most widely-cited source of data when it comes to the inclusion of API-related statistics [Pro13a]. That is 19 times the number available in 2010 [Pro11].
However, there are some things that APIs are not: [Jen15].

- **A piece of software:** Software is not an API, but it may use an API to provide services.
- **A user interface:** An API is not a user interface, but it may be build on top of that.
- **A server:** Server is not an API, but can host and provide it.

3.2 Elements of API Value Chain

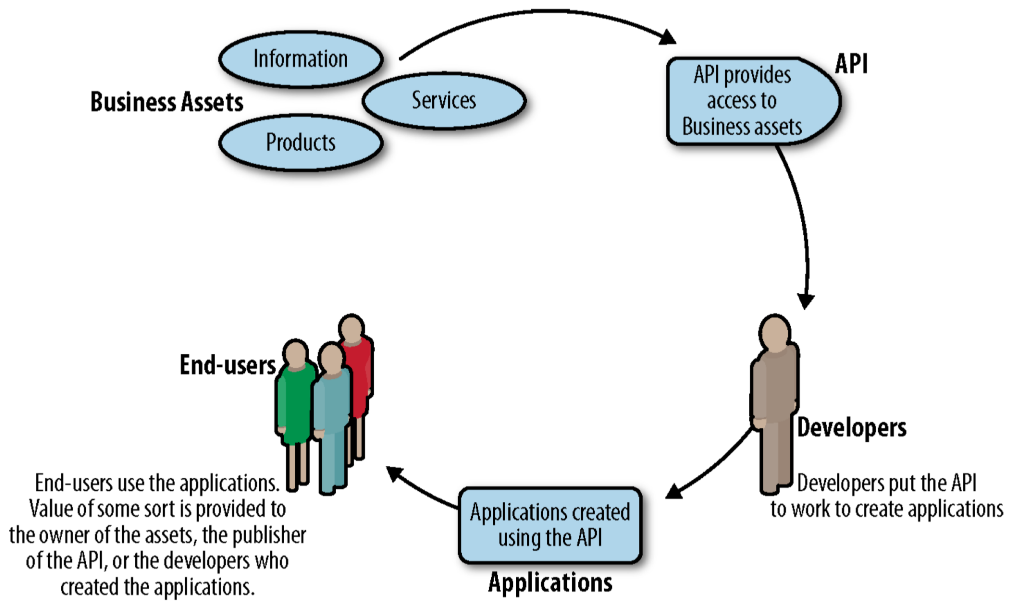

Information
Business Assets
Services
Products
API provides access to Business assets
API
End-users
Developers
Developers put the API to work to create applications
End-users use the applications. Value of some sort is provided to the owner of the assets, the publisher of the API, or the developers who created the applications.
Applications created using the API
Applications

Figure 3.1: The API value chain. [DW11]

When companies are managing the API, they should take into consideration business assets that it has, goals that want to accomplish and the interest of the stakeholders. [DW11]
The value chain starts which business assets, something that a business wants to allow others to use. **Business assets** can be a variety of things, like product catalogs, weather data, media content, maps, Twitter posts. If there is nothing in the business assets of a company, the API will not succeed. [DW11]
The next step is to create an API to expose these business assets. The API is designed and built by the **API provider**, and it is responsible for its availability. It must allocate the resources to API design, development, and maintenance. Furthermore, the API provider is responsible for documenting the API, so internal and external developers understand its functionalities. Moreover, creating incentives which push the developers to use the API which leads to awareness and increase of usage. [DW11]
Once the API is successfully published, and awareness is created, by evangelizing the API, hackathons, some traffic will hopefully put the API in production for building software applications. **Developers** are an excellent source of feedback, potential improvements, and new ideas. Therefore, the API provider should be in touch with them and provide support as much as possible. [DW11]
Furthermore, once created, the applications must find its way into the hands of end users. To find its way means that the application must be discoverable and reachable by the end users. Developers often use platforms like AppStore, PlayStore or other marketing channels to market their product and making them more discoverable. [DW11]
Finally, the value chain ends with the **end users**. End users are the ones who use the application, providing value to the developers, API provider and the owner of the business assets. [DW11]

3.3 From API to API Economy

APIs being part of products, are part of an agile business methodology, helping a company to create new revenue streams, by selling data on demand or by attracting investors, partners, producing an economy around itself, called **API Economy**.
The API Economy is the economy where companies expose their business assets or services in the form of APIs to third parties – partners, external developers – with the goal of unlocking additional business value, accelerating loyalty, and customer growth through the creation of new asset classes. [MB13, KH14]

3.4 Pioneers of API Economy

APIs have played a crucial role in the sector of Commerce, Social, Business, Cloud Computing, Mapping, Traveling, Weather and Mobile in the past two decades.
In the sector of commerce, for example, Salesforce.com which was officially launched in 2000 was a pioneer in the software-as-a-service area, being the first cloud provider. Using web-based APIs, they help customers integrating Salesforce services into their businesses, eliminating the need for expensive upfront costs, implementations that could take months and for the ongoing complexities of maintenance and upgrades. [Lan16]
In the year 2000, eBay launched the eBay API, along with the eBay Developers Program. At that time eBay stated:
"Our new API has tremendous potential to revolutionize the way people do business on eBay and increase the amount of business transacted on the site, by openly providing the tools that developers need to create applications based on eBay technology, we believe eBay will eventually be tightly woven into many existing sites as well as future e-commerce ventures."
The eBay API aimed to standardize how software applications integrated with eBay, making it easier for partners and developers to build services around the eBay ecosystem. [Lan16]
In 2006, Google launched Google Maps API, allowing developers to put Google Maps on their own sites using JavaScript, creating mashups with other data sources. It is reported by Google that more than a million active sites and software applications use the Maps API [Goo13].
In the same year, Facebook introduced Facebook Platform, which features the API at its core, allowing developers to access Facebook friends, photos, events and profile information. Immediately developers began to build social applications, games, and mashups with the Facebook API. [Lan16]
Later on, Twitter introduced Twitter API to the world, and in four years it had become the center of many desktops, mobile, web and business clients, engaging more users than the Twitter app itself. [Lan16]
In Cloud Computing field, Amazon saw the potential of a RESTfull approach to business and made APIs changing the way we compute now. In 2006 they launched Amazon S3 or Simple Storage Service, providing an interface that can be used to store and retrieve any amount of data, at any time from anywhere on the web. Initially, S3 was just an API, no web interface or mobile app. It was just a REST API allowing users fetching and putting data in cloud storage. Users were charged 0.15 dollars a gigabyte per month for storing the data in the cloud, generating new revenue streams. [Lan16]
In the traveling industry, Expedia opened a rich API to increase traffic and

to provide broader order values for its partners, including airlines and travel agencies. The API allows customers to access booking, photos, reviews directly on third-party websites and mobile applications. Based on that, Expedia generated 90 percent of its revenue through its API. [Lan16]
As these examples show, there are many ways APIs can accelerate growth for companies.

3.5 Growth of API Economy and its trends

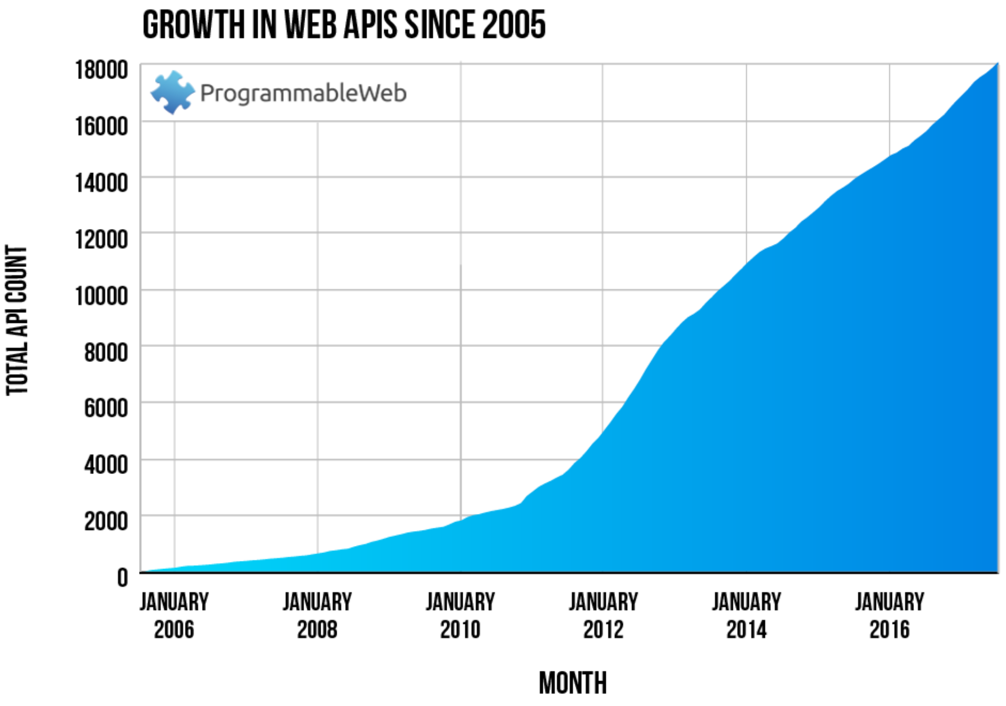


Figure 3.2: APIs added to ProgrammableWeb until August 2017. [Pro13b]

APIs are enjoying the mass exposure. The ProgrammableWeb directory at the first quarter of 2018 has more than 19,000 APIs, and this gives us a chance to look at what the data can tell us about the API Economy. Since the year 2000, APIs grow from a curiosity to a trend, and now to the point where APIs are core to many businesses, providing value to many companies and developers. But how does this growth compared with forecasts a few years ago? Is it sustainable? And why exactly has the space grown so rapidly? In January 2018, ProgrammableWeb saw more than 2000 new APIs added, and since 2014, an average of nearly 2000 APIs have been added per year.

Total new APIs added since 2014	8010
Average new APIs added yearly	1,982
Average new APIs added monthly	165

Note that these numbers may be deceptive because none of these take into account private or partner APIs that exist, which it is estimated to outnumber the public APIs. Some companies give access to their APIs through a premium account, making them less visible.[Pro13b]
Besides ProgrammableWeb data, I can also see statistics by some dominant APIs how they are growing in the recent years.

MailChimp which is a marketing automation platform and an email marketing service, in 2011 was grown from roughly 1.5 million calls per day to 9.5 million, peaked over 10 million and in 2016 they reached more than 29 million API calls. [Mai11]

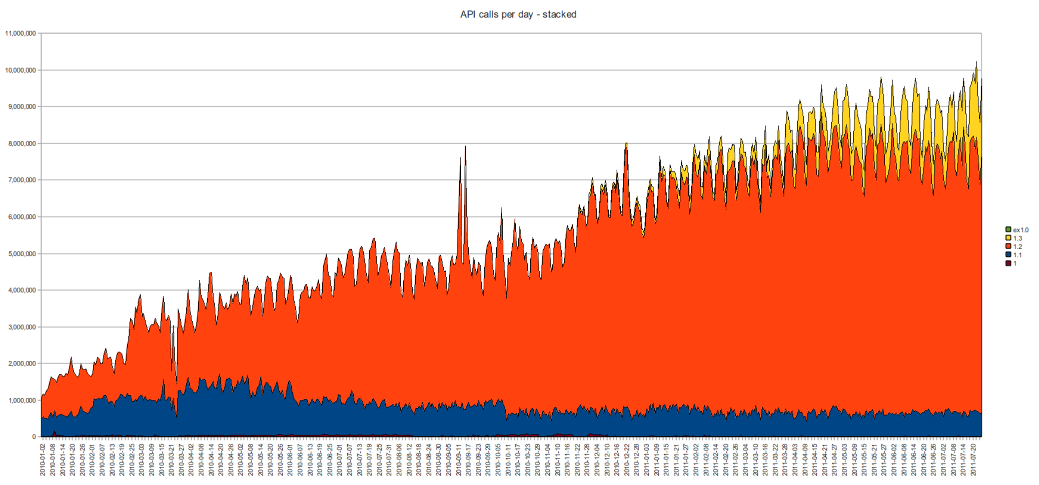


Figure 3.3: MailChimp API calls from 2010 to 2011.

Another case is Netflix which is an entertainment company streaming media and video-on-demand online that in 2008 launched the Netflix API, grow its API call in the beginning of 2011, 37 times more since the beginning. Netflix in 2014 discontinued its public API program and there are no fresh data to compare. [Net11]

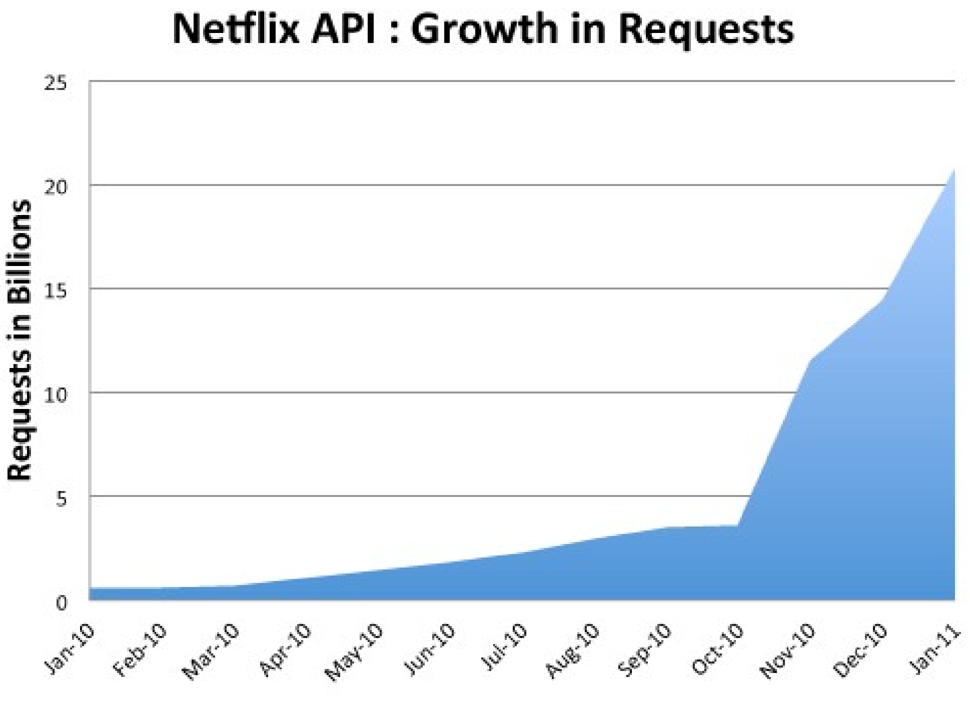


Figure 3.4: Growing the Netflix API by about 37 times in 13 months.

I could say that API Economy has been grown not because of a single sector, but by multiple factors emerging all on APIs to some degree. Based on the study by SmartBear, various areas impacted by the API revolution [Sma16].
Mobile, especially with the rise of Social Networks. As they appear to grow and evolve, APIs will play a vital role in the process of expansion via mobile devices. Internet of Things, because billions of devices are connected to APIs to fully unleash their extraordinary capabilities. Enterprise Integration is also an area in which APIs have a significant impact. For example, Salesforce allows integration through APIs only to Enterprise and Unlimited customers. They generate 50 percent of their revenue through APIs.

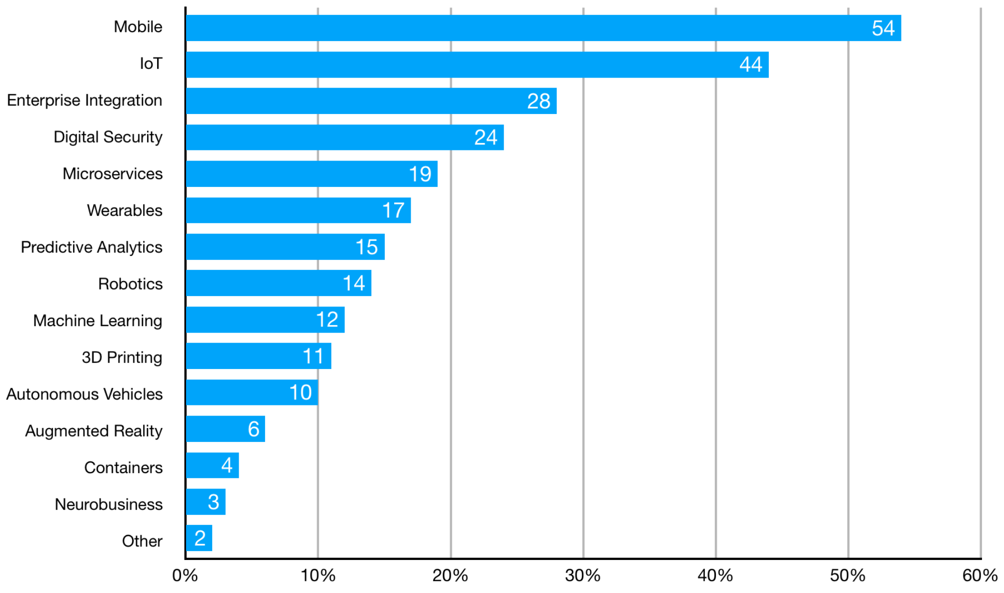


Figure 3.5: Technology areas that most APIs are growing

While many organizations have been developing and integrating with APIs for more than a decade, a significant number of them began providing APIs in the last couple of years. 42.1% of API providers have been providing/developing APIs for six years or more, while 51.5% began developing APIs in the last five years. 20% only began developing APIs within the last two years.

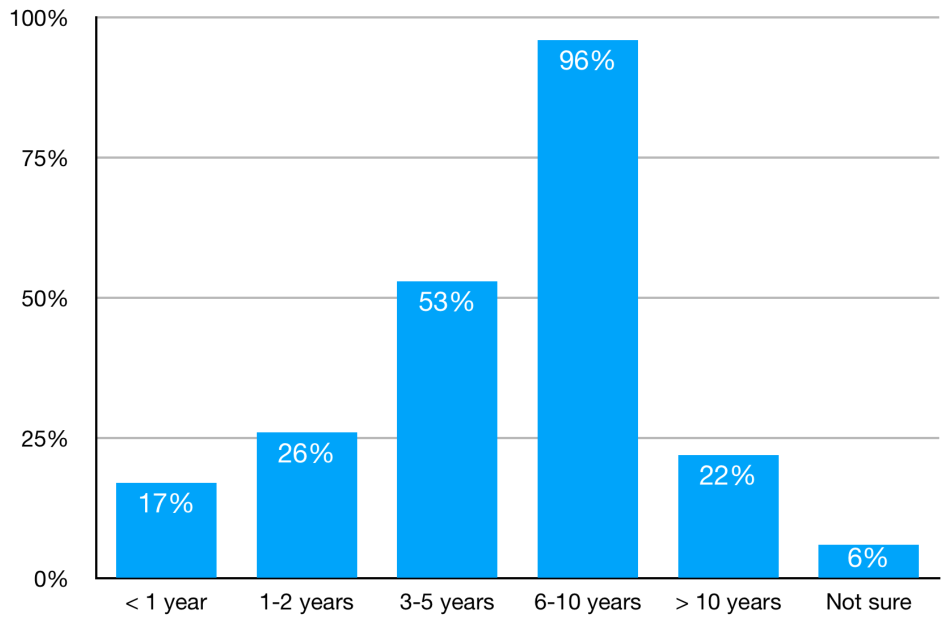


Figure 3.6: One in five API providers began providing APIs within the last two years

Moreover, looking to the future, two technologies are expected to drive the most growth for the API industry in the coming years, Mobile and the Internet of Things (IoT). Currently, Web and Mobile are top platforms supported by the APIs, while smaller in size compared to Web and Mobile. IoT is also being supported by 1 in 5 API Providers, showing that the newer technology is playing a sizable role in API Industry.

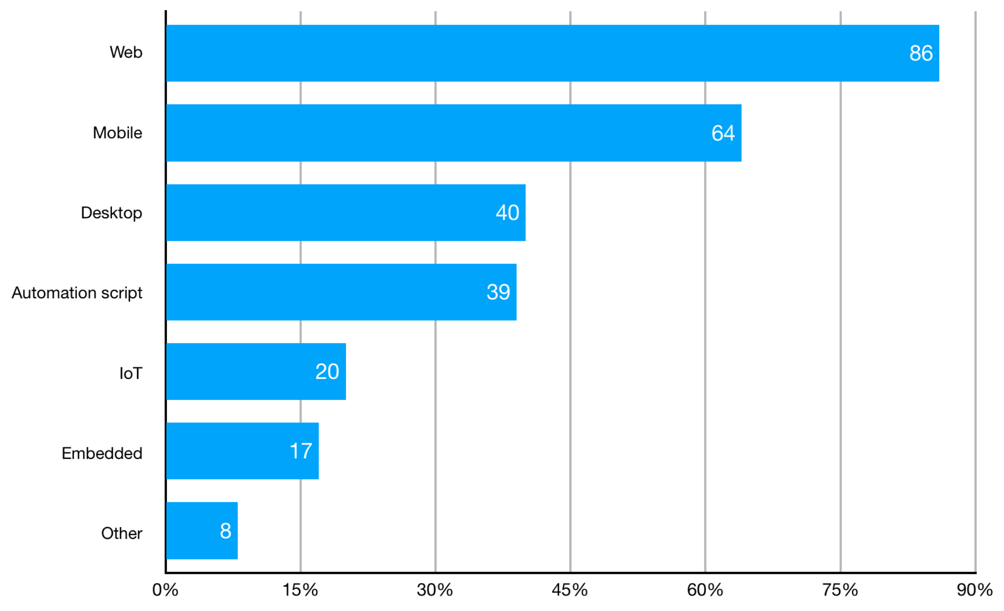


Figure 3.7: Nearly two-third of API providers support mobile.

3.6 Software, Platform and Infrastructure as a service

Within API space, cloud computing is accessible.
In this section, I will explain the cloud computing stack and its three layers SaaS, PaaS, and IaaS. In the world of IT, a stack is a series of interconnected systems that transport data between each other. The elements in a stack are called layers and are independent of each other. Moreover, the data in a stack flow from one end to another in sequential order without skipping any layer. [San16a]
For example, the OSI model is a stack where each layer handles a specific case and is independent of the others and is similar when it comes to cloud computing. Starting with the top layer, SaaS which stands for Software-as-a-service and is the most common form of service in the API space. In SaaS, the owner of the business assets licenses the use of the software and allows the other to use it based on a subscription model. A great example is Dropbox, which is a file hosting service that offers cloud storage, file synchronization, and personal cloud through a paid subscription model. [San16a]

Next, PaaS, which stands for Platform-as-a-Service, which provides a devel-

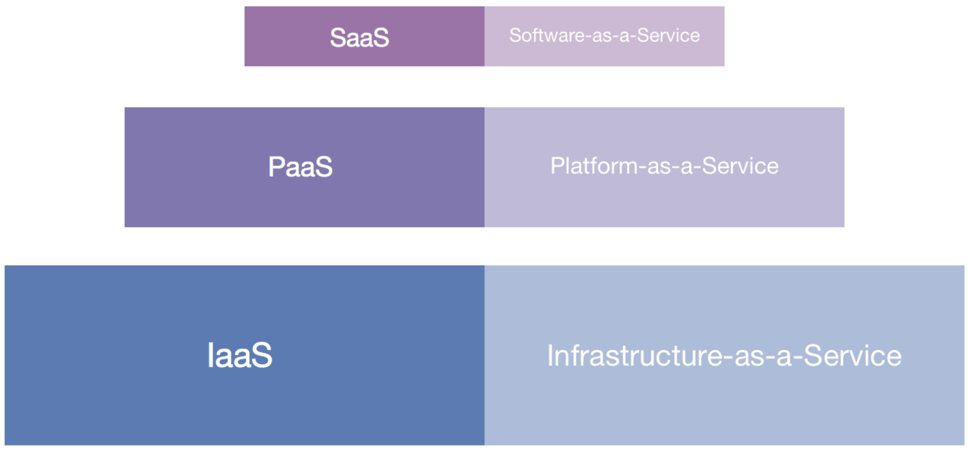


Figure 3.8: Cloud Computing Stack. [San16a]

opment platform remotely to a client. In comparison to SaaS, where remotely hosted applications are accessed via the API, PaaS supplies the entire development as a platform for the client. Offering servers, storage, networking, middleware, development tools, business intelligence and more.
For example, a company has services that manipulate data through a RESTfull API. As an API provider, you would need server space to store the data, processing power to compress the data, services to provide good compression and decompression algorithms. While using Microsoft Azure, you do not need to spend time and resources in building these things by yourself.
PaaS allows avoiding the expense and complexity of creating and managing the underlying application platform or the development of these tools. [San16a]

Also, IaaS, which stands for Infrastructure-as-a-Service and it's focused more on physical properties of the system rather than the platform or applications. IaaS makes it possible to avoid the expense and complexity of buying and managing your physical servers and other data center infrastructures. Whereas PaaS provides added benefit through a platform and SaaS though accessing the remote system to work within.
For example, a company needs storage space, fiber optics and extreme processing which is provided by Amazon Web Services (AWS) in remote. [San16a]

Chapter 4

API Lifecycle in the API Economy

Based on the information provided by the Systematic Literature Review, I am presenting an overview of the API lifecycle in Figure 4.1.

A crucial pre-step for the API provider, before deciding which business strategy will be used for the API, is validating why the company needs an API and who is going to use it. To enter in the API Economy network, the API provider should formulate answers for the following questions. *Why does the company needs an API?*, *What functions will the API accomplish?*, *What is the API value proposition?*, *Who is the audience?*, *Does and why my audience wants to consume an API?*, *Does the company has the resources required to develop and maintain the API?*
While answering these questions, the API provider measures the interest from target audience, performs market research, forecasts the trends while taking into account the internal resources. [Doe15]

1. **Planning Phase**
   An API is not just a part of the server that receives requests and sends responses. An API is also a business model. This is a new way of operation for companies about how are they going to engage with partners and developers. An API has to be seen as a product that needs to be developed with some business strategies in mind. Furthermore, it is essential to consider how the API will return the investments, boosting the revenue, and additionally the legal considerations for distributing the API data and protecting their work.

2. **Designing Phase**
   After planning and before choosing which technologies should be ap-

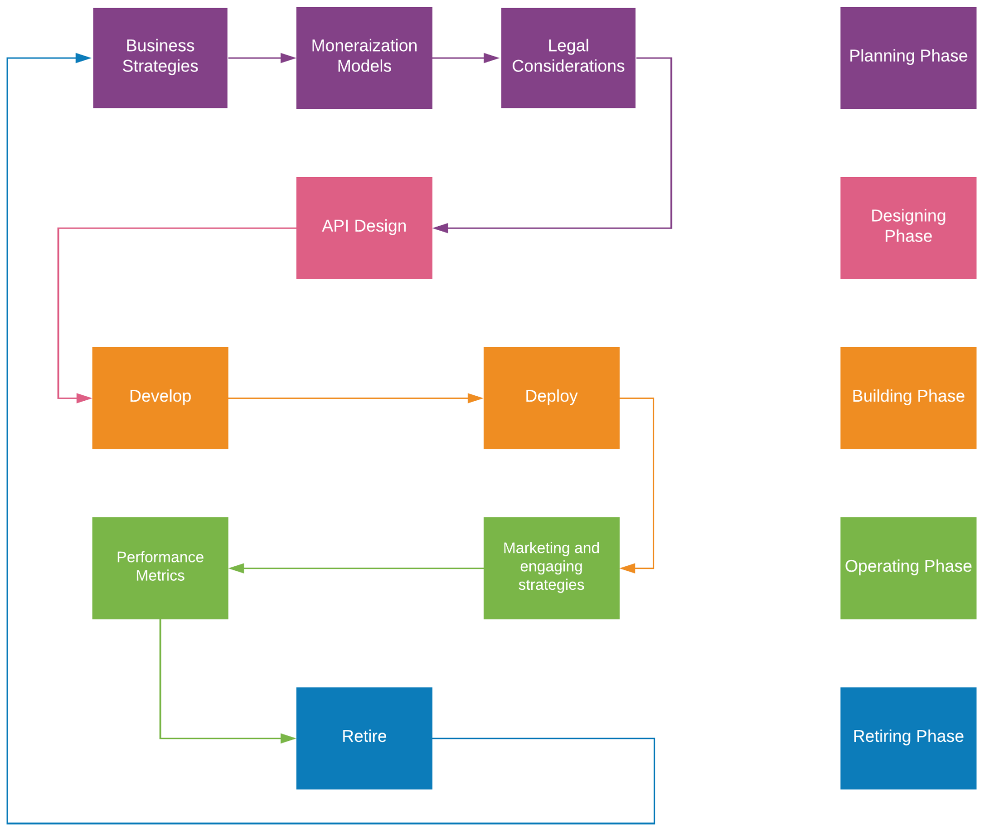


Figure 4.1: API lifecycle in the API Economy overview

plied to an API, a company should think about the API design. The design of an API is important and not easy. An API can be an advantage or a significant disadvantage for a company. A good API design improves the usability of it, resulting in higher adaption, higher user satisfaction and an overall better chance of success. While on the other side, a bad user experience while using the API may lead to a bad reputation. It is crucial to introduce and analyze different usability heuristics that can be implemented, giving a direction of how to craft an API.

3. **Building Phase**
A key action in the API lifecycle is to develop the API and bring it to life while deploying it. At this point, the company should have a well-defined API business strategies, together with monetization model, legal considerations, and design.

I will introduce and shortly analyze the methodologies and tools that can be used to develop, test and deploy an API.

4. **Operational Phase**
   After designing, building and deploying an API, a company should think how to market and engage developers with it. To obtain best possible results, marketing an API should be done across different channels. Each channel has a cost and a long way the cost of the channel has to be measured, understanding which channel fits best and which one should be abandoned. While engaging developers with the API, the developers can be a great source of feedback, helping the company to improve the API and hopefully, some traffic will come from the market. As a result, the company would need metrics for a variety of reasons, to measure the growth and success of the API.

5. **Retiring Phase**
   Retirement is part of API Economy lifecycle. In this phase, the owner of the business assets may decide to retire the API.
   Retirement can come from many things, for example, due to limited use, lack of third-party integrations or financial issues.

Each component of these phases will be analyzed in details in a dedicated chapter.

Part I

Planning Phase

Chapter 5

API Business Strategies

APIs are not just created to meet technical requirements. Every API is created with a business model behind. To ease the internal development, to attract developers, investors or partners.
Before launching or maybe before creating an API the company should think about the mindset of the API concerning the company. *Should the API be only for internal usage? Should the API be publicly used by the developers? Should we create the API to attract partners or investors?* or *maybe a mix of all of these?*
In this chapter, we will focus on API Business Strategies.
To begin, in a top level view, there are three main API models: **Private APIs** – internal or enterprise APIs, **Open APIs** – publicly released and ready for use by developers, and **Partner APIs** – integration between a business and their partners. Later on we will discuss on the monetization models that can give the **Return of Investment (ROI)** to a company.[Pro14]

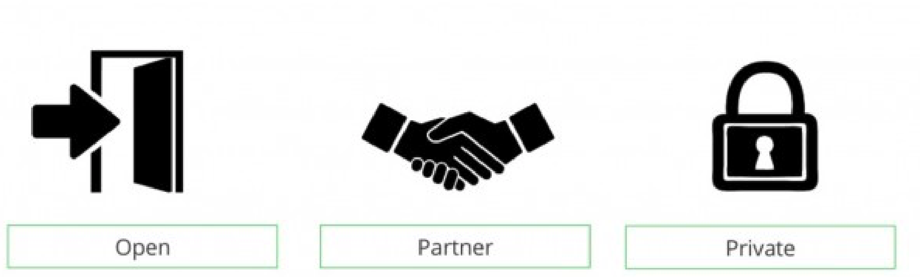


Figure 5.1: Three main API types [Pro14]

5.1 Private API model

Private API is to ensure that is tightly controlled and will restrict access only to internal users.
In private APIs, companies have no right or interest exposing their **business assets** outside the company or a tightly controlled domain.
In this case, the **API provider** is often the same as the owner of the business assets. **Developers** using the private API are often employees of the company or part of a restricted domain. **Applications** which are build on top of a private API can be used internally or publicly.
For example, private APIs can be used internally to create applications or to implement integration services. Also, it can be used as a way to more efficiently build apps for internal use in a company. For example, companies use their internal APIs to enable developers to build dashboards for distribution on tablet devices. [DW11]

5.1.1 Business Benefits of Private APIs

1. **Start the API Strategy**
   Some companies can start their API journey by being private first. They can experiment, learn, make mistakes and changes behind the doors, having less pressure from the community – investors, potential partners, and developers. In this way, the companies can learn from their mistakes and improve their strategy by being private first.

2. **Change internal structure of companies**
   While using private APIs, companies discover the opportunity to restructure their business to enable the *composable enterprise*. Composable enterprise is where people can support the changing demands of the organization by assembling and reassembling modular components. [Blo]

3. **Improve communication and collaboration:**
   A private API improves communication and collaboration between different teams and also their team members. A study by McKinsey [McK16] shows that companies can increase their productivity up to 25 percent by improving the internal collaboration and communication. One way to do that is the use of APIs. Using a private API allows for shared awareness of the internal data model. As the developers are working, communication will be more direct, and therefore they should be able to work more cohesively in a group or team.

4. **Accelerate Time-to-Market**
   Private APIs can accelerate Time-to-Market of new features and services. While having an API internal resources can be allocated more efficiently to create new features quickly.

5. **Internal analytics**
   Private APIs allow companies to identify better where there are problems in their workflow. Giving a better overview of what parts of our system might slow down.

An interesting use case is the Amazon internal APIs around 2002-2003. Amazon from an online bookseller company became the leader in eCommerce, IaaS and cloud computing. One of the secrets of becoming a leader is how the internal APIs were and are treated. Jeff Bezos the CEO of Amazon, issues that all teams will expose their data and functionality through application interfaces. All teams **must** communicate with each other through those interfaces, no direct linking, no shared-memory model. Furthermore, all the service interfaces, without exception, must be designed from the ground up to be externalizable. The team must plan and design to be able to expose the interface to developers in the outside world. If anyone who did not apply those rules was fired. Based on that, everyone worked for a couple of years, transforming Amazon to what it is nowadays. [Lan12]

5.1.2 Challenges and Disadvantages of a Private APIs

1. **Reverse Engineering**
   There is a fine line between public and private APIs, the security layer. Without appropriate security, the private API is a public API. There are several tools to deal with a private, undocumented API. Wireshark and Charles Proxy are two tools that can capture all the traffic send to. By using this kind of tools, the external (unauthorized) developers can analyze the traffic and interaction with the private API as if it were public. In this way, undesired data may be publicly available. [API15]

   A good example is Niantic, the company behind Pokemon GO. The developers or hackers, in this use case, reversed engineered their API, open source the solution and gave the end users the possibility of cheating the game.

2. **Being closed**
   While operating only internally inside a certain company, potential revenue streams may not be accessible because developers, investors, and potential partners are not aware that the API exists. [API15]

3. **Not focusing on usability**
   Often there is the case that API usability and code quality is not a priority in private APIs. By doing this, the productivity is decreased because is harder to understand the API's components and if API tends to go public or partner with another API, there is a high possibility to go first in technical debt. [API15]

5.2 Partner API model

In partner API model, an API provider is using its API to attract and integrate with potential partners.
This case is common among businesses operating in a business-to-business market. Use cases for partner API are automated invoicing, ordering and payment systems, products and price catalogs.
In partnering API model, the **API providers** are the owners of business assets. The **developers** that consume the API are employees of both companies, and the **end users** are the ones that are using the application if it is publicly released. In the partner environment, API development has to be conducted in a way that new features and changes do no break the existing integrations of the partner. [API15, Lan14],

5.2.1 Business benefits of Partner APIs

1. **Build trust with business partners**
   Providing a partner data so they can better manage their workflow via the API, is a great starting point to increase the trust.

2. **Increase profits**
   Using other company's API can help a company to increase their profits by buying data and services. Moreover, the ROI timeline is decreased faster because partners are willing to pay directly.

3. **Drive Innovation**
   While the partner is providing the resources to have a better workflow and focus directly on the business model without spending time on something that the partner is giving, can boost the productivity and

guide to innovation.
Amazon Web Services (AWS) is a good example that provides to their clients and partners computing power, database storage, content delivery. Also has the services to help the company build sophisticated applications with increased flexibility, scalability and reliability. Netflix on the other side does not have to spend time and resources caring about these services as long as they are built on top of AWS.

4. **Access new customer markets**
When collaborating with another company and sharing data with each other, one can access markets that couldn't access before without it. For example, Twitter collaborates with news sources to provide users' information quickly and reliably. In this was Twitter has access new markets, like news market, besides social networking.

5. **Identify commercial API opportunities**
In many cases, monitoring partner's use of business information can help identifying new commercial opportunities. Becoming apparent in some ways, helping to facilitate products faster.

One good example is Twitter, which established Twitter Partners Marketplace, helping businesses to thrive on Twitter. Twitter, focused on developing product partners with superior apps build on Twitter API and marketing partners that can deliver the services that Twitter needs [Rig15].

5.2.2 Challenges and Disadvantages of Partner APIs

1. **Increased bureaucracy**
Generally, while partnering, administrated procedures are more complicated because of negotiations and consultations between partners to jointly take decisions.

2. **Increased time for decision making**
When decisions inside a company may affect partners, it excessively increases the time for decision making because the partner's business model must be taken into account.

3. **Partnering when hiring is not possible**
Sometimes partnering with another company is not necessary, especially when the budget is not enough and the resources to hire new developers to integrate the functions that the potential partner may provide. This may be a killer from the start. In this way, decisions will be obligated by the partner/s under the partnership agreement.

In the example of Twitter, the challenge is that before becoming a partner you must swear to be a good partner, follow the guidelines, rules which for Twitter are acceptable [Rig15].

5.3 Public API model

A public API has been designed to be easily accessible to a broader audience of the web and software development. This means that the API can be used internally and externally. In this way, companies are turning themselves into service providers that enable other companies and free-agent developers to create the value that they want by using or paying for the data directly. This can stimulate new ideas and ease development and decrease its costs. [API15]
For example, DarkSky [1] is a company that is specialized in weather and forecasting visualizations, offers a public API which gives down to the minute weather predictions. Other companies that want to integrate some weather prediction services in their services can use DarkSky without putting much effort to create their weather forecasting service.

5.3.1 Business benefits of Public APIs

1. **New revenue streams**
   A company providing a public API can create new revenue streams if the developers are willing to pay for the data and services.

2. **Increase reach and traffic**
   Opening an API to the public can increase reach mostly by the third-party developers and other companies. If the API provides useful services, the company behind the API the has a good reputation inside this community. The APIs increase the user-satisfaction and overall better chance of success in the market among the competitors.

3. **Increase customer lifetime value**
   Public APIs build customer loyalty, in this case, developers. When an API is integrated into a 3rd party developer's value chain, they remain as customers for a long period. For many API users, a Public API integration can be a "set and forget" experience. Once the integration is established and any initial issues are resolved, the API can be used consistently in an end customer's workflow. This can create an ongoing revenue stream for the provider.

[1] Dark Sky Documentation: https://darksky.net/dev/docs

4. **Ease of development**
   Using Public APIs can help a company to ease the development and make better decision making work-flows. For example, a company can use Twilio public API for messaging services to reach their customers without spending time and resources in developing it.

5. **Improving without spending time on research:**
   While providing the API to the public, the developer community can give feedback on how to improve and what can be done better to the API. In this way, the developers are contributing to the API without the company spending time and money on experimentation and research.

5.3.2 Disadvantages and Challenges of Public APIs

While public APIs have many advantages, give companies new revenue streams and opportunities, but on the other hand, they also provide challenges and disadvantages that may bring a company to a bad reputation. [Mah14]

1. **Risk of exposing internal data**
   Going public leads to more attention also by hackers that want to gain access to private data through the API. The API is used as a gateway to gain access to internal data.

2. **ROI timeline is increased**
   In public APIs, ROI timeline can be longer compared to partner APIs, because of two main reasons. First, the API provider should establish an active community of developers integrating the API into new applications and services. Second, developers are looking to create new revenue streams first for themselves and then returning some part of it to the provider.

3. **Server overload and quality of service**
   It is a significant problem when the user waits too much to get the API's data even if they are valuable. If the API provider wants to increase the customer satisfaction and API usability, it should increase the number of servers capacity, efficiency and speed, compared to the case when the API is private or shared with a partner. Nowadays, thanks to cloud computing, the API providers are pushing their services to the cloud [Ped14].

4. **The need of a better API design quality**
   When releasing an API to the public, the provider should spend time

beforehand to improve the API quality. By API quality, it is meant that the API should be user-friendly, provide documentation, tests, examples how to use and what to expect by the API calls [Ped14].

5. **Conflict on interest**
   Conflict of interest happens when the third-party applications compete for the applications of the API provider. For example, if a company is a heavily ad-based application and not interested in charging developers for using the API, like Twitter, then yes it is an issue that someone can make a good application without ads [Ped14].

5.4 Wrap up

Some companies may start with a private API and then make them public. Others may start with a partner API, and then when they have learned how to implement and manage the API with some partners, they can move to the challenge of opening the API to the public. Also if feeling confident the company can start directly creating public APIs.
Even though private APIs are not considered part of the API Economy based on the API Economy definition, which is the economy where companies expose their (internal) business assets or services in the form of (Web) APIs to third parties with the goal of unlocking additional business value through the creation of new asset classes [MB13], it is good strategy to start with private API and then join the API Economy with a partner or public API.
Before taking the road path, it is important to know what are the benefits, challenges, disadvantages of each model and see if they fit with the company's business model.

Chapter 6

Monetization Models

Before developing an API, it is essential to consider how the API will return the investment, boosting the revenue. In this chapter will explain **how to monetize the API**.
According to Rob Zazueta, who is the director of a platform strategy at Intel, *"The success of an API program is measured by how well it moves a business toward its goal."* [Rig15, Ped14, Mus13]

6.1 Charge directly for API calls

One way to monetize the API is by charging directly for every API call. This model is easily measurable, if the API is returning some investments or not, but not necessarily successful. The model pays off when the data have the kind of value that developers or end users want to pay.
Before diving in, spend some time asking the developers – internal or/and external – or end users if they are willing to pay for the data and services and if they can, get an approximation on how much can they pay.

6.2 Tiered Licence

The most common one is the **freemium** model. Charging developers/end-users to use the API/data is strange because they do not know if it is worth spending money. Freemium model is a common solution to the API providers. They allow the developers/end-users to use the API/data for free up to a certain number of calls has been reached and after that they are automatically charged for API calls or cannot use the API/data until the fee is paid.
This is great for developers/end-users because it allows them to try something out before committing to it.

6.3 Charge based on the subscription model

The API provider offers access to a service or data for a weekly/monthly/yearly rate no matter the usage.

6.4 Charge based on units

In this model, the developer/end-users are charged every time they reach a certain number of API calls. Google Ad-words bills the end-users 0.25 euros per 1000 API calls.

6.5 Revenue share

In this model, the developer is paid by the provider when the API token that it is being used has reached a certain number of calls. Besides the developers, also the end-users can be paid. For example, YouTube pays the users if they reach a certain number of subscribers or Uber and Lyft pays the driver for every new rider.

6.6 Cost savings

In this model, the developer gets a discount when he/she puts more traffic than the threshold on the API. For example, Twilio's pricing model which is a mix of charge-directly, subscription, and cost savings. We would imagine for their enterprise deals looking like a tiered license might occur. Also, AWS's tiered pricing, per unit of infrastructure, or duration, charge-directly by the millions. Whatever method is chosen, consider mixing things up with a couple of different monetization methods to increase the chances for a faster ROI.

6.7 Premium upsell opportunity

This is a model used in the SaaS world a lot. Adding API access to a premium subscription of their services offers substantial incentives to upgrade to a higher option, as it allows end users to customize their experience and workflow more easily.

6.8 Content Acquisition

In this model, the end-user is paid directly and the provider indirectly. Companies that are based on user-generated content find ways to increase the quality of the content by rewarding the end-users every time they publish additional content. In this way, the end-users generate revenue streams for the provider indirectly. A good example is eBay Selling API. Third-party developers can easily use this API to list items programmatically, helping eBay increasing their revenue.

6.9 Content Syndication

In this model, the goal is to open the content as widely as possible putting more traffic from the consumers and more interaction with the product. Amazon is one example of a company that uses content syndication to distribute their inventory on as many third-party apps as possible. They do this with their Product Advertising API which provides programmatic access to their product selection-and-discovery functionality. Third-party developers can use this API to offer product search and detailed information, including product reviews and recommendations. As an added plus, developers also receive a share of the revenue generated from sales.

6.10 Wrap up

Nine monetization models can be categorized in 3 different groups, **developer pays**, **developer gets paid** and **indirect**.

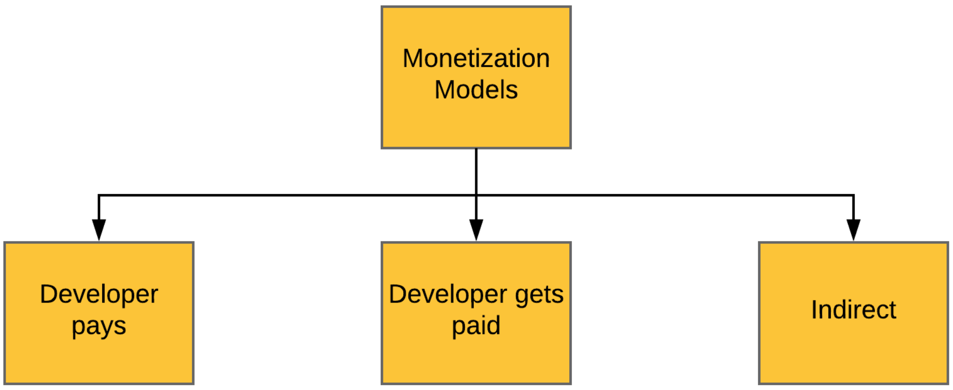


Figure 6.1: Monetization Categories

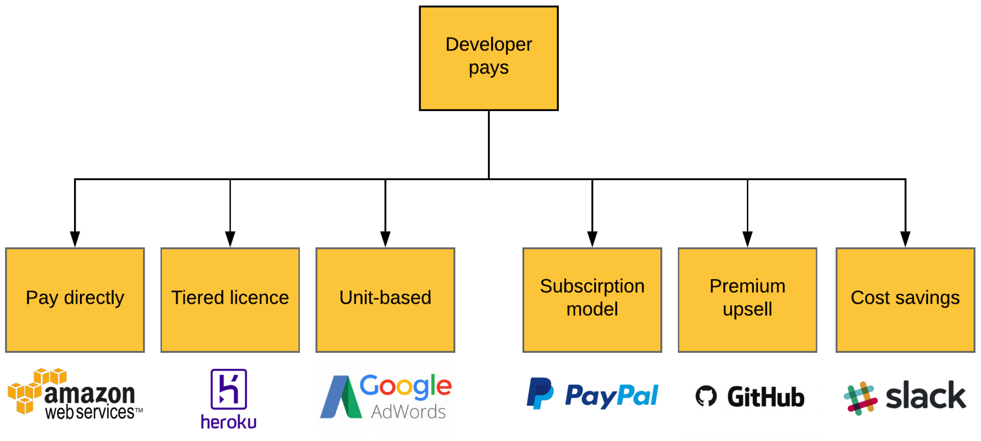


Figure 6.2: Developer pays

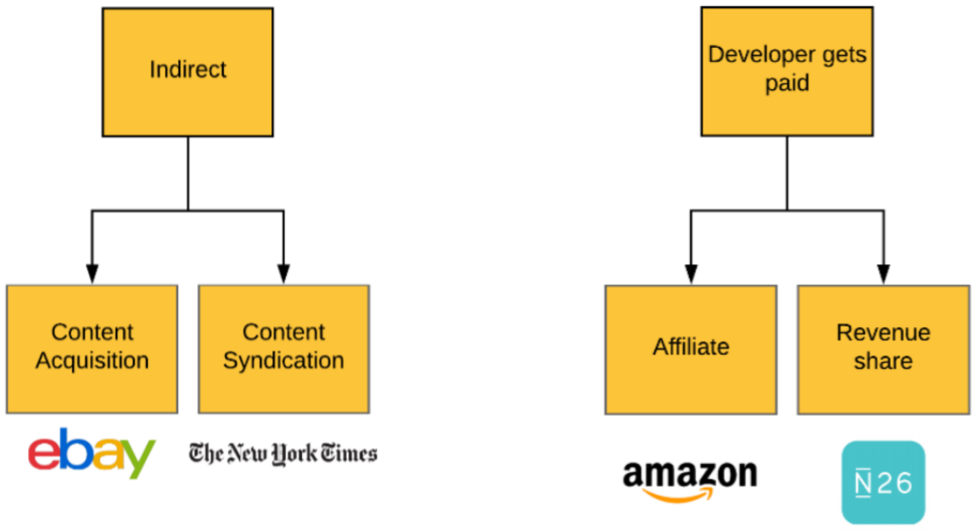


Figure 6.3: Developer gets paid and Indirect monetization model

Chapter 7

Legal considerations

Not understanding the ownership of the content an API provider is trying to publish publicly or between partners, like tweets, hypermedia content, market data and other information, can bring serious legal implications for the company.
Before designing an API take into consideration the rights for distributing the content to others and the rights that want to be granted to others who want to consume the API.
The first consideration is associated with legal rights and contractual relationships between a company and the others, while the other one is handled through direct contracts with partners and "Terms of use" contracts for the public.

7.1 Contracts and Terms of use

Terms of use differ for an API across target audiences because a public API and a partner API will have different contracts managing the relationships with the end users or developers.
Within the terms of use, it is important to incorporate trademarks, copyrights and branding requirements. For example, the Unsplash API, restrict the developers who consume the API from using "Unsplash" as the name of an application. [1]
Terms of use are often presented at the end users of the API and to developers that integrate the API into mashups and applications. When these are directed at the end users, the API infrastructure must include a mechanism to prevent API access until a particular end user is eligible for the terms of

[1]Unsplash API Guidelines: https://medium.com/unsplash/unsplash-api-guidelines-28e0216e6daa

use. Furthermore, when these change, the API infrastructure must ensure that the user is eligible for the new terms before letting them continue.
This sort of mechanism works well when combined with the OAuth, security model. To obtain an OAuth token, users usually provide their security credentials to the API provider using a web-based form. This login form asks the user for consent and not allow the login to succeed until it is granted. When the terms of service change, the API provider can revoke the OAuth tokens, causing them to automatically go through the authentication process, including granting consent, again. If a company has terms of use for a website, it is important to incorporate the API terms into the overarching terms applied to the site and/or other digital properties. [DW11]

7.2 Privacy policies

When a company handles sensitive data, privacy policies come into play. A privacy policy is a statement or legal document that discloses some or all the ways a party gathers, uses, discloses and manages client's data.[2]
The same principle applies when it comes to APIs, but is important that these policies be modified time by time to cover the growing needs that the API creates.
For example, if the privacy policy for a website ensures that any end user data will only be used on the website, then if the company plans to use the API for any mobile strategy, it is likely that the company will need to broaden the policy to allow the end user data to be used on all of the digital properties. On the other hand, if the company plans to offer a public API, it is possible that such user data will need to be restricted to the public audience. As a result, the privacy policy is not necessary to be changed. [DW11]

7.3 Data retention policies

Data retention policies are company's policies regarding the saving of data for regulatory or compliance purposes, or the disposes of it when no longer are needed. The policies emphasize how data or records need to be formatted, what storage devices or system to use and how to log these.[3]
Regarding APIs, data retention policies are designed to make sure that data used from the API do not grow stale, misinterpret, compromise the company and its users.

[2]Privacy policy: https://en.wikipedia.org/wiki/Privacy_policy
[3]Data retention policy: https://en.wikipedia.org/wiki/Data_retention#Data_retention_policy

For example, Facebook has a 24-hour data retention policy, meaning that developers are to retain user data 24 hours or less. Facebook wants to restrict retention to a relatively narrow window because its users are active in updating their content. Moreover, Facebook wants to restrict the time that this user data can be used to provide some degree of protection for their users. [DW11]

7.4 Attribution of Content and Branding

Brand awareness refers to the extent to which customers or end users can recall or recognize a brand, and it is a key business because it makes the end users or developers aware of the product.[4]
The same principle applies to APIs. When external developers use an API, the API provider must ensure that branding is shown and protected for every content in the API. [DW11]
The API provider should also show how the brand wants to be handled by the developers. For example, DarkSky which is a weather forecasting provider, states clearly in terms of service that applications or services which incorporates data should display the message “Powered by DarkSky”, together with the copyright symbol and logo. [5]

7.5 Service-Level Agreements

A Service-Level Agreement (SLA), is a contract between a service provider and the consumer that defines the level of service expected from the provider. SLAs specify what customers will receive, but not how the service is provided. [6]

SLAs are used in different ways, depending on the business strategy of the API. Regarding public API with a freemium monetization model, SLAs are quite weak. They make sure that the performance is excellent coupled with weak promises about service levels.
When it comes to partner APIs and public APIs where the consumer is paying, failing to have an SLA has greater consequences. SLAs need to include explicit documentation of remedies for outages, interruptions of service, security events, and the like. [DW11] Before starting the API Design, is impor-

[4]Brand awareness: https://en.wikipedia.org/wiki/Brand_awareness

[5]Dark Sky Terms: https://darksky.net/dev/docs/terms

[6]What is a Service Level Agreement: https://www.paloaltonetworks.com/cyberpedia/what-is-a-service-level-agreement-sla

tant to answer some fundamental questions regarding legal considerations. [DW11]

1. As an API provider, what are your rights regarding to the content you will provide though the API?
2. As an API provider, what are the rights that you can grant to consumers of the API?
3. As an API provider, are you talking into consideration other parts of the company that could impact the API?

7.6 Wrap up

Legal aspects should be taken in consideration to avoid implications for the company.
Terms of use are often presented at the end-users, developers, and are rules by which one must agree in order to use the API.
Furthermore, when companies handle sensitive data, privacy policies must be introduced. Privacy policies disclose all the ways a party gathers, uses and manages the client's data.
Additionally, data retention policies are designed to ensure that data which is consumed by the API are not misinterpreted and does not compromise the provider.
Moreover, branding is a key business, because it makes end-users to be aware of the product and the provider must make sure that branding policies are clear and how to brand wants to be handled by third-parties.
Last by not least, service-level agreements, which are contracts between the provider and consumer, defining the level of expected service, needs to be very specific in order to avoid misconception of the service and provide transparency to the consumers.

Part II

Design Phase

Chapter 8

API Design

To design a great API, it should be realized that APIs are in fact products and its customers are the developers. This means that when creating an excellent customer experience, it needs to create a great developer experience.
A good API design improves the usability, resulting in the higher adaption, higher customer satisfaction and in an overall better chance of success in the market. On the other side, a bad API design will lead to endless support calls and emails, followed by a bad reputation, which can make all the services unreliable.
During the year of 1995, Jakob Nielsen and Rolf Molich created a set of design heuristics for evaluating the usability of user interfaces. They are called **10 usability heuristics for user interface design**, are broad rules of thumb and not specific usability guidelines. [Nie95]

8.1 Visibility of system status

"*The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.*" — Nielsen [Nie95]

While using a software application, there are cases when the user can get easily confused when the software does not provide any relevant feedback. For example, when a user presses the button to upload an image and not getting any response from the application, would leave the user wondering if the button was pressed correctly or the image was uploaded with success or not.
The same principle should also be applied to APIs. Providing better information while calling the API, will help developers to understand what is

going on with the API, leading to a better development experience. Better information means the kind of information that is concise, understandable without going much into technical details.
From the developer's perspective, the interface is the concern, not what is going on in the underlying implementation. In an API takes longer to be executed successfully, the developer should get a meaningful, easy to understand the message.
In the case of Web APIs, the HTTP status codes provide concise, meaningful information about the system status. HTTP status codes are divided into five groups: **1xx, 2xx, 3xx, 4xx, 5xx**. [Who]

1. **1xx** include informational status codes that are intended to be used while the server continuous to process the request.

2. **2xx** include status codes that are used when an API request is successful, informing the caller that things went well.

3. **3xx** include redirection status codes that tells the caller to look for the requested resource somewhere else.

4. **4xx** include status codes that help the caller to understand what went wrong by his/her side.

5. **5xx** include status codes that shows the caller that something on the server side is not working as expected, no user fault.

Providing just 202 and 404 gives information that is understandable from the developers on what happened with the API call, but API designers can go beyond that and providing additional information in the message bodies, incorporating problem details. By increasing the system status information should increase the usability of the API.
When evaluating APIs answer the following the questions and look for cases where a lack of status visibility makes the interface harder to understand. [Mit]

1. Is it difficult to learn when something has gone wrong in the system?

2. Does the interface tell us the result of invocations and requests?

3. Should the system describe any relevant side-effects that may have occurred?

8.2 Match Between the System and the Real World

"*The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.*"
— Nielsen [Nie95]

A user-friendly user interface is that kind of interface that matches the user's model of the real world and interfaces that feel familiar are easier to learn and use.
When it comes to API usability, the user we care about is the developer that will use the interface. This means that the language and conventions of the API should match the user's world as closely as possible.
Mapping this heuristic to a concrete use case, let us mention some terms related to RESTfull APIs as part of the example [Hal]:

1. **Resource**, is an object or representation of something from the real world. This object has some attributes and a set of operations that can be applied on it. For example, "Movie", "Episode" are resources with a certain meaning in the real world and "create", "read", "update", "delete" are operations that can be applied on.

2. **Collection**, is a set of resources of the same type. For example, a collection of "Movies" or "Episodes".

3. **URL** (Uniform **Resource** Locator), is the path that a resource or collection of resources can be located and some operations can be attached on it.

Based on these terms some conventions that match the real world should be applied.
The resource should always be plural in the API endpoint, and if we want to access a resource, we can specify the operation and pass the ID in the URL.
The actions should come with a set of standard verbs that can be applied to noun-based resources available via APIs, like:

1. **GET** method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
   For example in RESTfull APIs, **/episodes/123/actors** returns list of all actions from episode with ID 123.

2. **POST** method requests that the server accepts the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database. For example in RESTfull APIs, **/episodes/123/actors** creates a new actor in episode with ID 123.

When evaluating APIs, answer the following questions to see if they match the real world examples. [Mit]

1. Does the language match the developer's real world?
2. Is the vocabulary understandable?
3. Does the API behaves like the APIs that developers are used to consume?

8.3 Consistency and Standards

"Users should not have to wonder whether different words, situations, or actions mean the same thing." — Nielsen [Nie95]

Consistency is an important part when it comes to user interfaces, and frequent changes can lead to unpleasant surprises and frustration, making the interface challenging to learn — as soon as developers got used to the previous version — and less usable.
The same thing also applies to APIs. When a developer learns how to use an API, really means learning the rules for using the interface of it. This includes every part of the API, from the error messages, authorization headers, query parameters of resources, resource attributes to the presentation style of the documentation.
Every frequent change in this scope establishes new rules that should be learned and tested by developers.
Of course, there are cases when change is inevitable and a good thing. When an API has reached a certain point, and it is expanding beyond its original intent, it is time to consider the next **version** of the API.
It is essential to consider the advantages and disadvantages of how to let the developers know about it. API versioning can have complex implications for developers and products where it is being used.

Once start taking things away or dramatically changing what's in place, it is time to consider another version. To do that, there are different ways:

1. **Traditional** way v+1 indicates significant changes to API consumption. It can tell a radical change, a minor change or just a bugfix. In each case, the developer should refactor the code in products to adapt to the new version because they are not usable anymore.

2. **One URI to rule all**, is another way for API versioning. The purpose is not to change to URI for every upgrade. If the API structure changes, resources are modified, the API is relaunched with the same URI. This way will push developers in code adaption and refactoring in the same way as in the Traditional way.

3. **Backward compatibility** is another way for API versioning. In this way API clients that still try to point to an old API should be informed to use the latest API version with **30x** HTTP status codes, which indicated redirection. The products that use the old version of the API will not be crashed at runtime, but developers are pushed to adopt new API changes because the old versions are not technically supported.

Before evaluating an API, look for cases where the developers would be surprised and confused by the way a particular part of the API looks differently from the conventions that are established in design.
When it comes to versioning, it is a multi-faced conversation and not just a technical problem. The new version needs documentation to transition successfully. Furthermore, the requirements for supporting multiple versions of an API can be very high regarding development support and management resources. However, in the end, it is essential to answer the following questions before taking any action:

1. Is the API consistent across its scope?

2. Is the vocabulary consistent and do the words have the same meaning?

3. Does the API implementation match the documentation?

4. Is it necessary to apply changes and version the API?

5. Do I have the resources to version and support old version of the API?

8.4 Error Prevention

"*Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.*" — Nielsen [Nie95]

Communicating errors is important, however, is even more important and even better to prevent users from making errors in the first place. The solution is to redesign a system to be less error-prone.
The term "user error" implies that is the fault of the user for having done something wrong, while actually, the design is the fault for making it too easy for the user to make an error. [Lau]
The same idea is also applicable to APIs. API errors include problems that can happen in client side, like typos, misspelling, syntax errors, also errors that can happen after the application is shipped for production. While preventing errors, the API provider can greatly improve usability.
To do so, as an API provider, provide examples how to use and test each API endpoint, document well what are the inputs for a resource and what outputs are expected back from it. Make sure that the design is not overly complex and does not have a high learning curve for developers to avoid misunderstandings and misuses.
When reviewing an API, look for these cases by answering the following questions. [Mit]

1. Does the documentation provide examples how to use the API?
2. Are the examples correct and not misleading?
3. Is the interface of the API overly complex and easier to be learned by the developers?
4. If complex, how can it be simplified?

8.5 Flexibility and efficiency of use

"*Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.*" — Nielsen [Nie95]

An interface which provides flexibility helps different types of users to be more efficient to the system. One common example is the keyboard shortcut for **copy-paste**. Probably for novice users, this is unseen, but after trying, it can be seen that remains an option that increases efficiency no matter in what context the user is using it.
Flexibility and efficiency of use are also applicable when it comes to APIs. An API that offers flexibility and efficiency of use to developers is easy to be consumed by novice developers and for more experienced ones, improves the efficiency. [Mit]
Cases that are worth mentioning are support of multiple API styles and formats, pagination, sorting, searching, filtering and different content types.

1. **Pagination**
   Many application that are powered by APIs, return pages and pages of data. Pagination provides efficiency when the dataset is too large, is divided into portions of data called pages, which improves the performance and is easier to handle performance. There are three primary pagination types: [Doe18]

   (a) **Limit/Offset**, uses a limit in the number of results returned and the position of the first result(offset) to determine the result returned.

URL	Description
.../episodes	Returns the first 50 episodes (the default limit is 50).
.../episodes?limit=10	Returns the first 10 episodes.
.../episodes?offset=20&limit=80	Returns episodes in range 20..80.
.../episodes?offset=10	Returns defects in range 11..61 (the default number of the returned episodes is 50).

Table 8.1: Example: Limit/Offset Pagination in RESTfull APIs

(b) **Page/Page Size**, uses Page Size to indicate how many results should be returned, while the Page tells which set of results to return.

URL	Description
.../episodes?page=1	Returns the episodes in page 1.
.../episodes?page=3& per_page=20	Returns the episodes in page 2 and 20 episodes in that page.

Table 8.2: Example: Page/Page Size Pagination in RESTfull APIs

(c) **Cursor based** paging returns a token value or URL that is provided with the following call to retrieve the next set of results. The cursor token is usually combined with a value to determine the max results to return.

URL	Description
.../episodes?since_id=12&max_id=34	Returns the episodes that are between these 2 IDs.

Table 8.3: Example: Cursor Based Pagination in RESTfull APIs

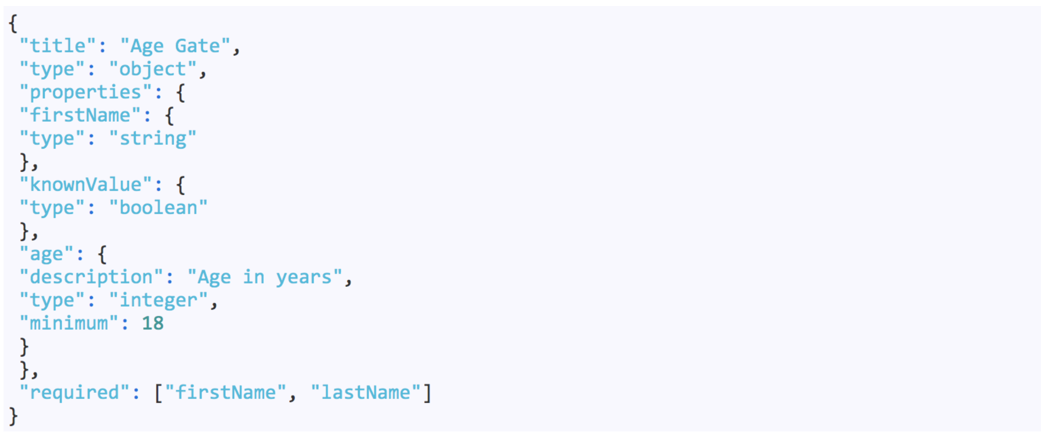
2. **Searching & Filtering**
   Exposing resources through a single URI can lead the client requesting a significant amount of data, when sometimes only a part of the data is necessary and usefully.
   For example, a developer working on an application to include a list of episodes that have a rating higher than a certain threshold. Without the possibility to search/filter the URI, the developer has to implement all the logic in client side, which results to be inefficient and inconvenient. Furthermore, the network bandwidth is wasted, together with the processing power of the server where the API is hosted.
   Instead, the API can allow the developer to pass some parameters in the query string of the URI such as "**.../episodes?minRating=9.5**". In this way the result is transmitted faster and in a more efficient way to the user. [mic16]

3. **Sorting**
   Similar principle as for searching & filtering applies to sorting. Developing sorting algorithms in client side takes extra time for the developers.

Moreover, the developers should know how to design an efficient algorithm, because when it comes to big data, the algorithms may suffer. Furthermore, as I said it before, the processing power of the server is wasted for not being taken into consideration.
Instead allow the client to pass parameters in the query string of the URL like "**.../episodes?sort=name&rating**"

4. **Content Types**
   When designing an API, one of the most important things to consider is data formats or saying it differently, content types.
   The bridge between the client and the content must be created in a way that is usable by the client and recognizable by the server.
   An API with well-crafted architecture and implementation is not usable if the supported content types are limited, that is why is essential to support many of them.
   Content types can be separated into four general categories. Direct data, feed data and database data formats. [San16b]

   (a) **Direct data formats** are designed to handle data between machines. Three most common direct data formats are **JSON** (JavaScript Object Notation), **XML** (EXtensible Markup Language) and **YAML** (Yet Another Markup Language)



```
{
 "title": "Age Gate",
 "type": "object",
 "properties": {
 "firstName": {
 "type": "string"
 },
 "knownValue": {
 "type": "boolean"
 },
 "age": {
 "description": "Age in years",
 "type": "integer",
 "minimum": 18
 }
 },
 "required": ["firstName", "lastName"]
}
```

Figure 8.1: JSON data format example

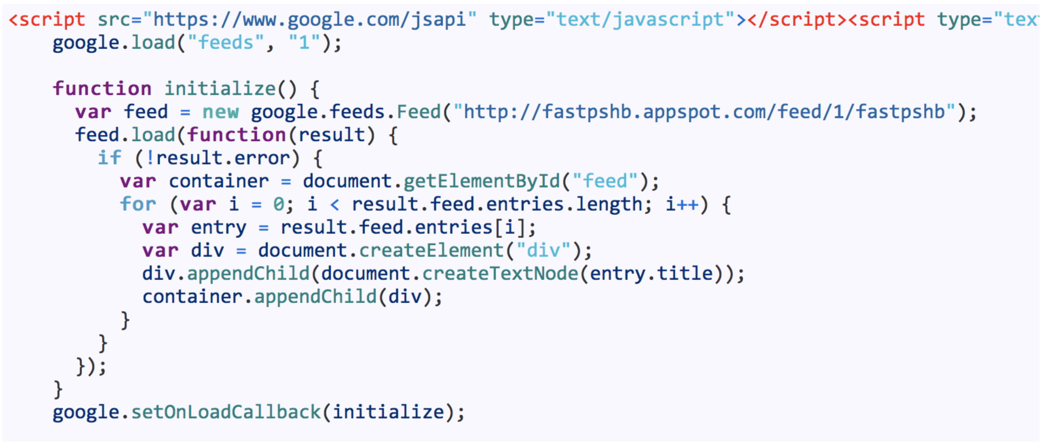
```
<xs:element name="user" type="lettertype"></xs:element>

<xs:complextype name="lettertype" mixed="true">
  <xs:sequence>
    <xs:element name="usertype" type="xs:string"></xs:element>
    <xs:element name="userage" type="xs:positiveInteger"></xs:element>
    <xs:element name="regdate" type="xs:date"></xs:element>
  </xs:sequence>
</xs:complextype>
```

Figure 8.2: XML data format example

```
# failover url
url_403: /
#url_403: http://example.org/underage

# snippet definition
snippet_enter: /templates/verified.html.twig
snippet_exit:  /templates/underageexit.html.twig
snippet_403: /templates/validate.html.twig

# minimium age config
min_age: 18

# container variable
width: 300
height: 300

# container bg
overlay: '#ffffff'
```

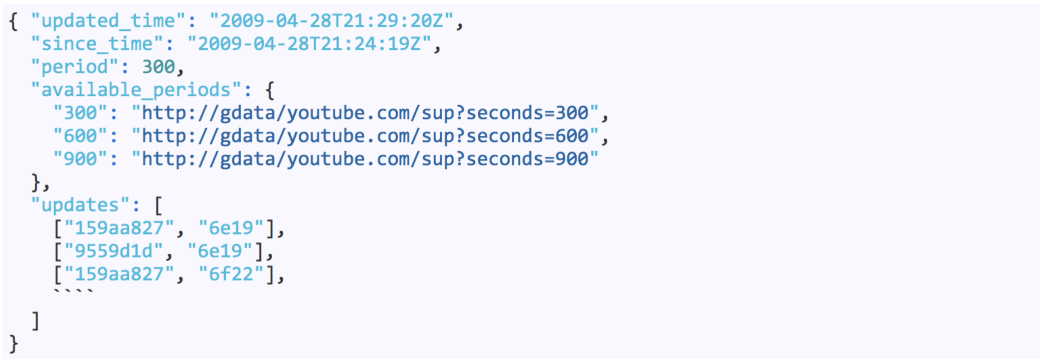
Figure 8.3: YAML data format example

(b) **Feed data formats** are tied more to user entity rather than machine usability. Formats in this category are: **RSS** (Rich Site Summary), **Atom** and **SUP** (Simple Update Protocol). Feed data formats are typically used to serialize updates from different servers, sites or front-end interfaces and alert users in these changes.



```
<script src="https://www.google.com/jsapi" type="text/javascript"></script><script type="tex
    google.load("feeds", "1");

    function initialize() {
      var feed = new google.feeds.Feed("http://fastpshb.appspot.com/feed/1/fastpshb");
      feed.load(function(result) {
        if (!result.error) {
          var container = document.getElementById("feed");
          for (var i = 0; i < result.feed.entries.length; i++) {
            var entry = result.feed.entries[i];
            var div = document.createElement("div");
            div.appendChild(document.createTextNode(entry.title));
            container.appendChild(div);
          }
        }
      });
    }
    google.setOnLoadCallback(initialize);
```

Figure 8.4: Atom data format example



~~~
{ "updated_time": "2009-04-28T21:29:20Z",
  "since_time": "2009-04-28T21:24:19Z",
  "period": 300,
  "available_periods": {
    "300": "http://gdata/youtube.com/sup?seconds=300",
    "600": "http://gdata/youtube.com/sup?seconds=600",
    "900": "http://gdata/youtube.com/sup?seconds=900"
  },
  "updates": [
    ["159aa827", "6e19"],
    ["9559d1d", "6e19"],
    ["159aa827", "6f22"],
    ````

  ]
}
~~~

Figure 8.5: SUP data format example

(c) **Database data formats** are used to handle communication between users and databases. While direct data formats generate data upon request, database data formats take generated data and archive it for later use. Formats in this category are **CSV** and **SQL.**

While an API provider could spend a lot from the budget in development to expand the content type support, it can end up with an API that requires more support than usual and can lead to failure if certain content types are not used. This is a risk of supporting many content types in an API.

When reviewing an API, look for these cases by answering the following questions. [Mit]

1. Does the API provides shortcuts to boost the efficiency like pagination, searching & filtering and sorting?
2. Does the API supports at least the most popular content types like JSON, XML and CSV?
3. Are there opportunities to optimize any repetitive or unnecessary steps?

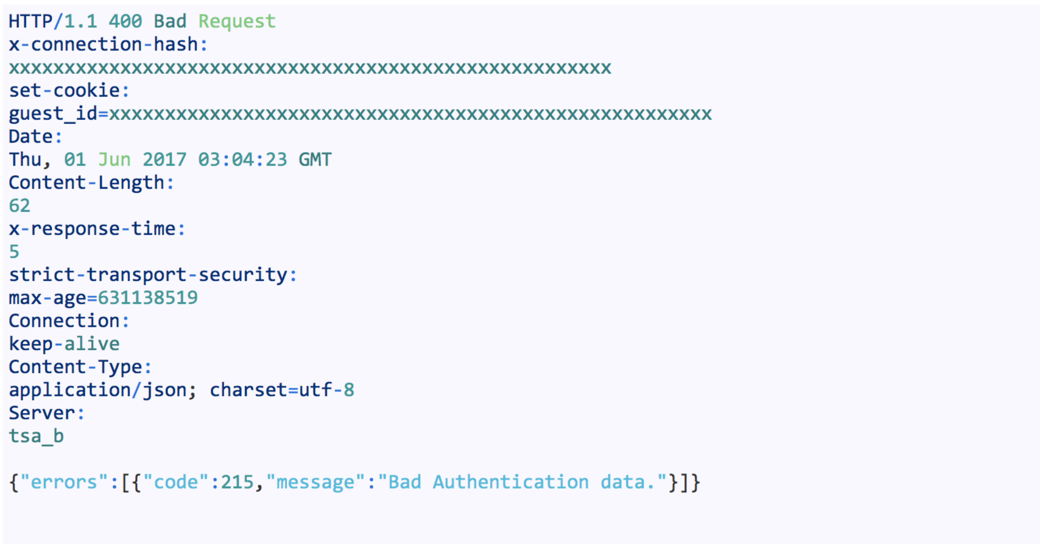
8.6 Help Users Recognize, Diagnose, and Recover from Errors

"*Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.*" — Nielsen [Nie95]

API error messages have two audiences: **The machine-driven client** that acts programmably on what it receives and **the developer** that is responsible for writing applications that use the API. In this section we care about error messages that target the developers. [Mit]
Based on Error Message Guidelines by Nielsen, good error messages should include:

1. **Explicit indication** that something has wrong, because when developers make mistakes and get not feedback from the API, they are confused and completely lost.
2. **Human readable language** instead of codes, otherwise the developers will not understand well what exactly went wrong.
3. **Polite messages**, that does not blame the developers doing something wrong.
4. **Precise descriptions**, that describe the exact problem that happened while calling an API endpoint, rather that vague messages like "syntax error".
5. **Constructive advice** in error messages that helps the developers how to fix the problem.

A great example is Twitter API, which provides descriptive error messages that helps the developers to understand the problem and how to recover from it. [San17a]



```
HTTP/1.1 400 Bad Request
x-connection-hash:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
set-cookie:
guest_id=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Date:
Thu, 01 Jun 2017 03:04:23 GMT
Content-Length:
62
x-response-time:
5
strict-transport-security:
max-age=631138519
Connection:
keep-alive
Content-Type:
application/json; charset=utf-8
Server:
tsa_b

{"errors":[{"code":215,"message":"Bad Authentication data."}]}
```

Figure 8.6: Twitter API error message

First, by looking at the data, we can see that we have submitted a **400 Bad Request**, which leads us to know that the problem is in our request.
We can see, however, that we are receiving a unique error code that Twitter has called it “215” with an attached message “Bad Authentication Data”. This error message gives an explicit indication that something went wrong. Also, it is human readable and understandable. Furthermore, it does not blame the developer for doing something wrong. It is precise because it describes the problem that occurred. Moreover, it gives constructive advice, because our error lies in the fact that we did not pass any authentication data.
When reviewing an API, look for these cases by answering the following questions. [Mit]

1. Is the error information correct?

2. Is the machine readable information provided?

3. Does it describe the error in a way that the human use can understand it?

4. Is enough information provided to correct the error?

5. Does the error messages conform to “Error Message Guidelines” by Nielsen?

8.7 Help and Documentation

“*Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.*” — Nielsen [Nie95]

API documentation is a technical deliverable containing instructions about how to effectively use and integrate an API to a certain software application and is the main interface through which the new developers interact with the interface.
Developers, novice or experienced, use the documentation to initially understand the functionalities of the API. Furthermore, due to errors that can happen while execution, the documentation can guide the developer to handle these errors.
In one hand, having proper API documentation, allows new developers to understand the functionalities behind an API solution effectively. However, on the other hand, not having good documentation could overly complex the API.
To start with, it is important to assume nothing about the developer experience when writing the documentation. The language should be easy to understand without distracting the readers. Moreover, by eliminating assumptions, will force the team to classify terminologies and therefore relate concepts more clearly to developers.
To continue with, most documentation describes the functionality of an API in a static and theoretical way. This kind of documentation merely changes unless new features are implemented. In this way, the functionality is explained without having the possibility to experiment directly.
To make the documentation dynamic rather than static is by introducing **sandboxes**. Sandboxing is a controlled, known environment where developers can test API calls and their functionalities against the resources [San17b]. Sandboxing plays a major role in API documentation by giving the possibility to truly learn the API rather than memorizing it.
In static documentation, the developer has to memorize or constantly reference the documentation.

With time, the developers will gain more knowledge and better understand the API, but the learning curve from beginning to end is high.
In a sandbox, the developer manipulates data in real time and experiment with the API endpoints in a way that is not possible in the static documentation. A great example of sandboxing merged with documentation is the GraphQL API Explorer from GitHub and Slack API Documentation. Both of them allows the developer to test functionality within the sandbox environment and demonstrates the ease by which a developer can learn through experimentation rather than memorization.
Despite having or not sandbox implementation in the documentation, there are four essentials that every API should have in its documentation. [San18]

1. **Authentication scheme**
   Authentication is the process of recognizing a user's identity that tends to use an interface, and it is the key to manage the user's rights over the interface.[1]
   An API may offer many actions to modify the data, and some of them may require some specific rights which are specified during the authentication process.
   There are many authentication schemes, but the most common are:

   (a) **HTTP Basic Auth** is an approach in which the user provides a username and password in HTTP Header when making the request.

   (b) **API Keys** is another approach in which a unique generated value is assigned to each first time user, meaning that the user is known or unknown for the system.

   (c) **OAuth** is an approach that combines authentication and authorization in its process. When the user logs in into a system, the system will request authentication in a token form. The authentication server where the request is forwarded will reject or allow the request. Such a token will give the user the rights across the interface. Figure8.7.

   Failure to document the authentication method transparently, will not only make the discovery of API data much more difficult, but it can also push the first time developers not to use the API because of the complexity that the documentation perceives.

[1]Authentication: https://searchsecurity.techtarget.com/definition/authentication

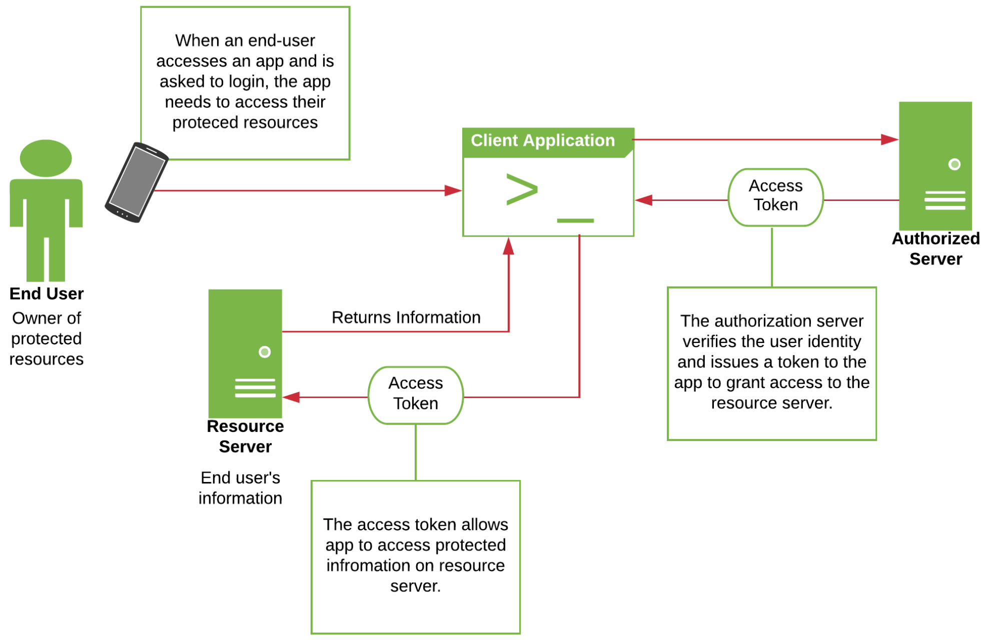

When an end-user accesses an app and is asked to login, the app needs to access their proteced resources
Client Application
Access Token
Authorized Server
End User
Owner of protected resources
Returns Information
The authorization server verifies the user identity and issues a token to the app to grant access to the resource server.
Access Token
Resource Server
End user's information
The access token allows app to access protected infromation on resource server.

Figure 8.7: OAuth authorization process.

2. **HTTP Call Methods**
   Developers interact with API resources with various HTTP call methods and documenting which and how methods are used for every resource is very important for the API usability. If the HTTP call methods are not part of the documentation, it is not transparent for the developer how to access the resource's data.

3. **Requests and Examples**
   Just stating in the documentation that the API call exist means nothing unless it is explained its functionality, its limitations and how to access the data.
   This not only will help the developer to understand the API in greater level but will also allow contextualizing the functions of the API as a whole. Not showing example API calls that turn the technical information into actionable data, makes the documentation incomplete.
   Furthermore, examples help the developer to onboard faster and learn more thoroughly about the resource functionality by doing.
   Also, providing examples can help the developer to recover and prevent errors while looking at their failed requests and compare them with what it should be.

4. **Expected Responses**
   Having the expected response for every resource, part of the documentation is critical. When the developer builds a software application on top of an API, it is valuable to know how the resource's data look like before calling and manipulating them.

API documentation can be evaluated in the same way as any technical writing. Before evaluating it, try to answer the following questions:

1. How well the documentation map the problems that a developer will try to solve?

2. Is the language and vocabulary understandable?

3. Does it provide information that helps the developer to understand the functionalities behind the system?

4. Does it include the four basic essentials that were explained above?

Even though the Nielsen's heuristics give strong fundamentals about API design and usability, a specific API architecture can help the developers to solve problems efficiently and provide more opportunities. For that, I want to write about two most popular API architectures: REST and GraphQL, a comparison between them and why companies like Github are changing their API from REST to GraphQL.

8.8 REST vs. GraphQL

REST is an architectural concept, with no official set of tools, no specification and is designed to decouple an API from the client. REST focuses on making APIs last for decades, instead of optimizing the performance.
GraphQL is a query language, with a list of specifications and tools, designed to operate over a single endpoint. GraphQL focuses on optimization for performance and flexibility.

REST and GraphQL, both of these architectures exchange data over HTTP, but GraphQL has some small changes to make a big difference to the developer experience of building and consuming APIs.
The core part of REST is the resource. The URL identifies the resource, and the developer interacts with it through built-in HTTP verbs.
Due to this nature, there are some issues with REST that GraphQL fixes.[San17c]

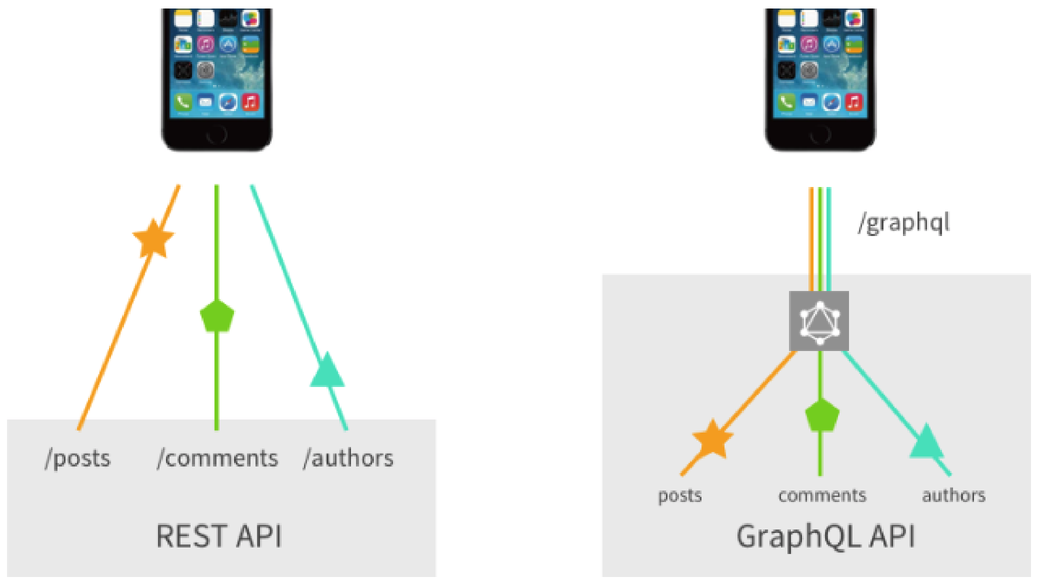


Figure 8.8: Interpretation of fetching resources with multiple REST round trips vs. one GraphQL request [Stu17]

1. **Round trips and repeat times**
   In most cases, in production APIs, the problem is that these resources are complicated and relational to each other.
   Fetching these requires round trips between the client and the server to retrieve the data. Also, in some cases repeated trips are necessary. This is a big downside for REST because as more as relational the resource is, the slower is to retrieve its data.
   One of the benefits of GraphQL over REST is the fact that GraphQL has fewer round trips than REST, which makes it more efficient.
   GraphQL has a single endpoint, and it unifies the data that would exist in multiple endpoints, like in REST, and creates packages.
   By packing the data, its delivery is more efficient and the amount of resources for round trips and repeated times is decreased.

2. **Over/Under-fetching**
   Over-fetching happens when more data is fetched than necessary from the client, while under-fetching happens when not enough data is fetched upon request.
   For example, in REST, if a developer requested a list of products and their prices, the result would have product descriptions and images. If prices were quoted in another resource, the developer would have to

make another request to retrieve them.
GraphQL is build to take care of this problem. In there, the developer can make data query, specifying what to retrieve. These results are delivered through shifting the data definition in client side, rather than in server side, like in REST.
Over/Under-fetching is a common problem in REST because the server defines which data to return, while in GraphQL the server declares the available data and the client specifies what should be returned.

3. **Dynamic typing and poor metadata**
   REST suffers from dynamic typing.
   Dynamic typing means that the values are checked during executing, and a poorly typed operation might cause a halt in the system or signal an error during runtime. On the other side, GraphQL uses static typing, which means that the operations are checked before being executed, and the system might reject before it starts.

To summarize, both REST and GraphQL are just ways to call functions over a network. If building a REST API is familiar, implementing a GraphQL API will not feel much different. However, GraphQL has some advantages because it calls several resources without multiple round trips, requests and retrieves the exact data that is needed and moreover the operations are checked during compile time.

8.9 The GitHub GraphQL API

One and a half years ago, GitHub announced the fourth version of their API, which supports GraphQL. GitHub changes the API focus from REST to GraphQL to solve two problems.
The first problem to solve was scalability. The REST API that now is part of version 3 was responsible for over 60 percent of the requests.
GitHub uses a lot of hypermedia content, and because of that, hypermedia navigation requires a client to repeatedly communicate with the server to get all the information that it needs. API responses were filled with all sort of URL hints in the JSON responses to help users navigate through the API, and this was not a flexible way.
Sometimes, it was required to do two or three API calls to create a complete view of a resource. Furthermore, API responses sent too much data and did not include the data that the users needed.

The second problem was encountered when GitHub was auditing its endpoints. Then wanted to collect meta-information about endpoints, they wanted to be smart about how the resources could be paginated, and they wanted to ensure type safety. Because of these reasons, they support GraphQL in their fourth API version. [Tea16]

8.10 Wrap up

From 10 heuristics, only 7 of them are applicable to API design, because 3 others: “User control and freedom”, “Recognize rather than recall”, “Aesthetic and minimal design” are relevant only to user interface design.
Even though these heuristics are 23 years old, they are still relevant today, also applicable in the API design, giving **strong fundamentals** no matter the architecture.
By working through each heuristic, think critically an in details about the API and furthermore, how can be improved.
In spite of the fact these heuristics give strong fundamentals, a specific API architecture can help developers to solve problems efficiently, providing feasibilities and more opportunities. Based on this reason, I did a comparison between REST and GraphQL and why companies are migrating to GraphQL.

Part III

Build Phase

Chapter 9

API Driven Technologies

More than a decade ago, software applications were physically shipped in CD-ROMs and being unused until the user's initial installation. Nowadays, most of the code is shipped over the Internet, which means that continuous software update is not only achievable but also expected by developer and end-users and DevOps play a big role in this shift.
DevOps stands for Development and Operations, which is a combination of teams, tools and best practices that increase the ability of a company to deliver services and applications faster. By delivering faster, companies can innovate and improve products faster. The quicker the release of new features and bug fixes, the faster the response to the customers' needs.
Furthermore, with the help of DevOps, a company ensures the quality of service/application updates and infrastructure changes. In this way, a company can reliably deliver at a faster pace while maintaining positive feedback from developer and end-users. [Ama18]
Transitioning to DevOps requires changing the internal culture and mindset. DevOps helps because it removes the barriers between two teams, development, and operations, in which they work together to optimize the productivity of developers and reliability of operations. Both teams communicate more frequently, increase productivity and improve the quality of services they provide. Besides development and operational teams, quality assurance and security teams may also become tightly integrated with those teams.
Companies should use a DevOps model, regardless of their internal structure and have teams that view the development and infrastructure lifecycle as part of their responsibilities. [Ama18]

9.1 DevOps practices

Automating and streamlining the software development and infrastructure processes are key principles that help organizations innovating faster. Accomplishing these practices proper tooling is needed. [Ama18]

1. **Frequent with small updates**
   One practice is to perform very frequent with small updates, to innovate faster for the customers. Frequent with small updates makes each deployment less risky. In this way, teams identify bugs faster because the last deployment that caused the error can be identified easily and fixed.

2. **Microservice Architecture**
   Companies can also use microservice architecture to make their applications more flexible. Microservice architecture decouples large, complex systems into simple, independent project. Applications are broken into many individual components with each component being responsible for a single purpose and operating autonomously of its peer components. This architecture reduces the coordinator overhead of updating the applications. For example, Netflix uses microservice architecture to manage their services all around the globe.

3. **Continuous Integration**
   Continuous Integration is a DevOps development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests run. The key goals of continuous integration are to find and address bugs quickly, improving software quality and reducing the time it takes to validate and release new software updates.

4. **Continuous Delivery**
   Continuous delivery is a DevOps development practice where code changes are automatically built, tested, and prepared for a release to production. With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment. There can be multiple, parallel test stages before production deployment. In the last step, the developer approves the update to production when they are ready. This is different from continuous deployment, where the push to production happens automatically without explicit approval.

9.2 Types of Continuous Integration

9.2.1 Traditional CI

Jenkins was the first to gain widespread adaption. Build by Sun Microsystems in Java and open sources in 2004. In Jenkins developers can set up a workflow to get their code from Subversion, Git, Mercurial and trigger code builds, run tests and deploy every time builds and tests are successfully passed.
Initially, Jenkins was meant to support Java, but developers created plugins to support many other languages. Jenkins is only available as on-premise, but some cloud providers have created solutions based on Jenkins.
Similar to Jenkins, is TeamCity by JetBrains, which is another on-premise Java-based tool and Bamboo by Atlassian which can be either on-premise or hosted on Amazon EC2.
Tests in a CI workflow have evolved. In the beginning, only unit tests were included. Testing objects and components in isolation. Later on, automated integration testing was born, in which individual software modules are combined and tested as groups. [API16]

9.2.2 Cloud CI

Many development teams were tired of self-hosting their CI system. Configuring it often proved to be consuming in time, money and resources.
TravisCI is a cloud-hosted CI service build on top of the GitHub API. Developers build any projects that are hosted on GitHub. Travis starts building and running tests for every commit or merge request.
Development teams use this kind of cloud-based solutions to no longer face the problem and do extra work for managing those tools. [API16]

Nowadays, most developers would not consider working on a project without setting up a CI workflow. APIs are becoming a cornerstone of internet communication, and its stability is crucial and caring about, so CI is playing and will play an important role in the API development.

9.3 API Management Services

Unaware that there are service providers that can help plan, deploy, launch and manage the API infrastructure, companies often choose to do all the

work by themselves.
Below I will do a short analysis of some vendors in the API Management market. The analysis below is based on the study "Magic Quadrant for Full Life Cycle API Management" by Gartner. [PM16]

9.3.1 API Design Services

1. **Apiary** offers a platform for the API design life cycle and is the sponsor of API Blueprint, an open source markdown language for describing APIs. It is offerings mainly apply to the design, implementation stages and also includes versioning functionality, but does not directly offers an API Gateway or developer portal. Moreover, the platform encourages and guides API providers though iterative API design lifecycle. However, if the API is already designed, Apiary is not relevant to the case. [PM16]

2. **Oracle** has a long tradition in the infrastructure market. To fulfill API Management requirements in its client base, Oracle had an agreement for Vordel API Gateway which is integrated and sold as Oracle API Gateway. Its full product line is generally market as Oracle API Platform Cloud Service which offers API Manager, API Manager Cloud Service, Oracle API Catalog, and Oracle Communications Services Gatekeeper. However, Oracle's offering is still young, and its functionality in policy management and the developers' portal need to be enriched to be comparable with competing products in API management. [PM16]

9.3.2 API Development Services

Continuous Integration has become the cornerstone of software development and is a fundamental part of agile development.
In the API community, continuous integration tools are part of two groups. In the first group, there are tools like Jenkins, TravisCI, CircleCI, Bitrise, BuddyBuild and Bamboo that help drive the **build processes**, allowing automated builds and **execution of tests**.
The second group contains tools that help companies implement continuous integration for the API they are creating. These tools execute tests and other performance metrics related to the running processes. In this way, this kind of tools allows companies to introduce a higher degree of automation.
Even without these tools, testing is possible using just shell scripts and cURL, but this takes more time and effort which may not be available. [API16]

9.3.3 First group of CI Tools

1. **Jenkins**
   Jenkins is an open source platform for project automation. It works as a standalone continuous integration server with a web interface that can be used to configure the server quickly. Jenkins supports common use cases like building projects, execution of tests, bug detection, code analysis and project deployment. Moreover, Jenkins is cost-free. [Sta17]

2. **Travis**
   Travis is a continuous integration platform that automates the process of software testing and deployment of applications. Travis CI is free for all open source projects hosted on the GitHub and for the first 100 builds. There are a few pricing plans that can be chosen and the main difference is in the number of concurrent builds it can run. [Sta17]

3. **Circle**

   Circle is a continuous integration platform that helps developers to release their code through build automation, test execution and deployment processes. Circle CI is different from other tools by the way they offer services. The main pricing block is the container. One container is offered for free, and in there it can be added as many projects as needed, but once it starts getting more containers and the levels of paralyzations per build are chosen, you start paying. [Sta17]

4. **Bamboo**
   Bamboo is a CI platform that is used by software developers to automate the process of release management for applications and services. Bamboo gives developers a way to automate their builds, tests, can also support different stacks like Docker and AWS and works with different programming languages. Bamboo is not offered for free. It costs 10 euros for small teams and 800 euros for growing teams. [Sta17]

9.3.4 Second group of CI Tools

1. **Abao**
   Abao is a Node.js package that allows developers to test their Restful API Modeling Language - RAML - against server where the API is located. To test the API, "abao" command should be executed on

command line tool, passing the RAML file and API endpoints as parameters. [API16]

2. **Dredd**
   Apiary introduced continuous integration tools into their platform, enabling developers to automatically test their API with test definitions generated from their API Blueprint description. [API16]

3. **Postman**
   Postman is an API testing client tool with support from importing API descriptions in RAML and Swagger. API developers can create tests using Javascript. The Javascript code can be put into collections from being executed by automated tests. [API16]

4. **Swagger Diff**
   Swagger Diff is a command line tool to test backward compatibility between two versions of a Swagger description for a given API. On finding a breaking change, the tool will return an error, providing details of the diff that highlights the source of the incompatibility. [API16]

The continuous integration tools that help developers to maintain and keep the API consistent easily, highlight the fact that these tools are a growing presence in the API Economy, with a mixture of open source and commercial solutions. The mentioned tools are not intended to be a product review, rather more highlighting the tools that are highly used by the API community.

9.3.5 API Deployment Services

It is a significant problem when users wait for an extended period to get the API's data even if they are valuable. If the API provider wants to increase the customer satisfaction and API usability, it should increase the number of servers capacity, efficiency and speed, compared to the case when the API is private or shared with a partner. Nowadays, thanks to cloud computing, the API providers are deploying their services to the cloud without setting up the servers and maintaining them for their API by themselves. [Ped14].
API deployment technologies let companies to securely deploy their APIs to API Management platforms like AWS API Gateway, AWS Lambda, or Microsoft Azure.

1. **AWS API Gateway** is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. The AWS Management Console gives the possibility to create an API that acts as a front door for applications to access data, business logic, or functionality from your back-end services, such as workloads running on Amazon Elastic Compute Cloud, code running on AWS Lambda, or any Web application. Amazon API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. Amazon API Gateway has no minimum fees or startup costs. Pay only for the received API calls and the amount of data transferred out. [1]

2. **AWS Lambda** is a zero-administration compute platform for back-end web developers that run your code for you in the AWS cloud and provides you with a fine-grained pricing structure. AWS Lambda runs back-end code on its own AWS compute fleet of Amazon Elastic Compute Cloud (Amazon EC2) instances across multiple Availability Zones in a region, which provides the high availability, security, performance, and scalability of the AWS infrastructure. [2]

3. **Microsoft Azure** is a cloud computing platform and infrastructure created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed datacenters. It provides both PaaS and IaaS services and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems [3]

9.3.6 Full API Lifecycle Services

1. **Apigee** is one of the leaders in the API Management and a loud advocate of APIs. Apigee Edge is the code Apigee management platform which is available both on-premises and in the cloud. Apigee also provides Apigee Insights, Edge Exchange, and IoT server. Furthermore, Apigee provides cloud-based bot detection and protection. For example, if an API is found under automated attack, it can be proactively defended. [PM16]

[1]Amazon API Gateway: https://aws.amazon.com/api-gateway/
[2]AWS Lambda: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
[3]Microsoft Azure: https://azure.microsoft.com/en-us/

2. **Axway**, also another leader in API Management. It provides a full range offer like API Gateway which includes a policy design tool, also API Manager which contains the API catalog and registry of API consumers. However, Axway harnesses its power as long as its customers need to go through a learning curve which may struggle the developers to keep up to pace with the requirements. [PM16]

3. **CA Technologies** comes from the acquisition of Layer 7. CA has CA Live API creator that allows developers to build APIs quickly. Besides this, CA offers API Gateway, Mobile API Gateway and API Developer portal. The solution is offered on-premises, cloud, and hybrid. However, CA should improve the ability to innovate faster to new market trends. [PM16]

4. **Cloud Elements** is a niche player when it comes to API Management, which focuses on solving the problem of organizing existing APIs for easy consumption. The platform allows APIs to be grouped into categories called API Hubs which combines multiple API into a single interface. The solution is available on-cloud and on-premises. Furthermore, it provides advanced security and analytics. One advantage of Cloud Elements is the pricing model. The platform is based on the number of API consumers, not on the number of API calls, which means that customers can take advantage of unlimited API calls. [PM16]

5. **Dell Boomi** is a business unit that is part of Dell, which provides API Management. It supports two deployment models, a cloud model for when all endpoints are a cloud-based and on-premises model for when any of the endpoints are within a corporate framework. However, the solution lacks to provide an external developer portal. Also, Dell Boomi lacks support for helping organizations plan their overall API strategy. [PM16]

6. **IBM's API Management** offering is IBM API Connect. It is available both on-premises and on-cloud. IBM API Connect is available at three levels: Essentials which is free for developers, Professional, which is for small business and prices according to API calls and Enterprise, which is higher call volumes and has the flexibility to shift from on-premises to on-cloud. Part of Professional and Enterprise are the two API Gateways: Micro Gateway based on Node.js and Enterprise API Gateway. IBM has an established and influential market position, a solid customer base and it has worldwide support and diversified geographical strategies. [PM16]

7. **Mashape**, was found in 2010 and launched the API Marketplace in 2012 with the goal of becoming a giant API Hub and marketplace. In 2015 Mashape introduced Kong, which initially was built to secure and manage the marketplace. Today it consists of tens of thousands of private and public APIs, as well as many active developers, generating billions of request per month. The solution is offered both on-cloud and on-premises. [PM16]

8. **MuleSoft** is a provider of cloud and on-premises integrations. The solution is made up by API Designer, API Developer Portal, API Analytics and API Gateway. MuleSoft has shown effective marketing and leadership in the API Management market. By MuleSoft there is an assumption that is an entirely open source, and a lot of API related components are, but API Gateway and developer's portal are not. [PM16]

9. **RedHat** has become a highly successful open source provider who gives a broad spectrum of solutions like IaaS, PaaS, visualizations, containers, and tools to manage all these. [PM16]
   Recently RedHat acquired 3scale which is a mature API Management provider. The solutions can be on-premises or on-cloud. 3scale gives RedHat a fully functional API Management platform, including also a developer's portal. Furthermore, it brings RedHat a strong foundation of understating the market needs and customer's main priorities. [PM16]

9.4 Wrap up

During this phase, companies should develop and bring the API into life while deploying it. I wrote shortly about the DevOps methodology and its practices. Additionally, about types of continuous integration and API management services.
The providers behind the management services are mentioned because they market any subset of API management, both on-premises, and on-cloud. Furthermore, they provide a comprehensive, general purpose offering a full lifecycle API management that covers API lifecycle stages – designing, developing and deploying. Moreover, the providers include a developer portal, and all mentioned providers generate revenue of at least 10 million euros per year. All of these can be used to develop and deploy an API.

Part IV

Operational Phase

Chapter 10

Marketing & Engagement Strategies

10.1 Evangelizing the API

Evangelizing is one of the methods to market an API. Usually, evangelizing is done by gathering and meeting developers that are interested and have knowledge about building APIs, sharing stories and experiences about how an API can be used. The API evangelists are used to promote the API thought social media and their blogs. Usually, evangelizing an API is cost-free, especially if the provider does it, but there are cases when it can become costly when a group of API evangelists is paid to review the API.

10.2 Hackathons

Hackathons are events, typically lasting several days, in which a large number of people meet to engage in collaborative computer programming [1]. When it comes to API promotion, Hackathons are a popular method, because API providers have indisposition a large number of developers that can use and test the API for certain prototypes. Also, the developers can furthermore promote the API through word of mouth to other developers. In this way, though a hackathon, an API provider have higher changes for discoverability and API usage. Usually, hackathons are geared toward a specific industry and are location-based, allowing a certain company to target specific demographics.

[1]Hackathon: https://en.wikipedia.org/wiki/Hackathon

10.3 Integrations

Another method to gain attraction is by attaching the API to mashups. It can be done by creating integrations that will make the API easily available and consumable by developers who use other products. In this case, the cost will mostly be associated with software development activities and also about maintaining and supporting the integrations after it goes live.

10.4 Engaging developers

Developers have lots of options to choose in the API market. ProgrammableWeb in May 2018 lists around 20.000 public APIs. To reach developers, the API provider should understand what motivates them to choose the API among the others.
First of all, the API provider must provide a solid API and communicate how the API is different from the others that might be similar across different channels.
Furthermore, developers should be able to try the API with some successful results, quickly. Even if the API requires a formal partnership, the API provider can still provide at least some operations that do not require registration and partnership agreements. For example, Twitter offers without any registration to fetch public tweets.
Derek Gottfried from New York Times says that while lowering the bar for the developer to get involved with the API, the better responses the API provider can get. For example, New York Times offers code samples and mashup examples highlighting the best use case for the API. [DW11]
While offering this kind of feasibilities, the API provider gets traction from the developer community.
Moreover, developers expect clear and accurate documentation for the API. Good documentation can be viewed as strong point why the API should be used.
In addition to API documentation, the API provider can offer a developer gallery with examples of application and code that can be copied and pasted from the sample gallery into the development application.
Also, one of the most effective things the API provider can do is to drive awareness of the API where the developers hang out. ProgrammableWeb, GitHub, and other developer sites often have listings, app galleries, and forums where the provider can post the news about the API.
Once these communities have been identified, the API provider must identify the developers that have the respect and admiration of their peer and the

ability to rally other developers. While spending some time in these communities, it will not be hard to identify these developers. They are the ones that post the most and respond often. While reaching them, the API provider gets help to improve the product with their programming experience. [DW11]

10.5 API Branding

Branding is a marketing practice of creating a name and meaning to market that identifies and differentiates a product from another [2]. The brand communicates its business value and reflects the value of a company. The same principle can be applied to API Branding. Based on a research by IBM Institute of Business Value [Cor16], a strong pattern for branding includes consistency, clarity, and constancy with developers and end customers in mind.

1. **Consistency**
   Companies that thrive in the API Economy emphasizes the brand consistency. They stress that their APIs retains the original look, feel and functionality across other companies products and services. If a company makes changes to the overall look, it is likely to break down and being lost.

2. **Clarity**
   Communities are a great source where API evangelists get the information about a certain API and the company behind it. So many API producers must be clear in communicating API features and functions while making it easy to find them among APIs. Some things that API providers should address are the functionalities provided by the API, the technical requirements and the steps to get started.

3. **Constancy** API developers perceptions and expectations about an API and the capabilities that an API can offer should be aligned together. Brand constancy is important due to the risk of brand delusion. So, to sustain brand constancy, an API should be easy to understand, have precise terms and conditions regarding expected levels of service and acceptable forms of usage.

[2]Branding: https://www.entrepreneur.com/encyclopedia/branding

10.6 Wrap up

Having an API does not matter if nobody is using it. To obtain best possible results, marketing about the API should be done across different channels. Each channel has a cost and a long way the API provider should measure the cost of it, understanding which channel fits best and which one should be abandoned.

Chapter 11

Metrics

Hopefully, some traffic will come from the market. As a result, the API provider would need metrics for a variety of reasons to measure the growth, success and performance of the API. The more the API provider knows, the better it will be able to see and demonstrate what is going on.
There any many types of metrics that can be captured analyzed and used to develop more effective APIs.

11.1 Usage metrics

Usage metrics are metrics that help the API provider to understand how the consumers, in this case, developers and end users interact with products, services, content which is served by the API. [DW11]
For example, The Guardian, which is a media company, provides a public API, through which developer can create a variety of applications where end users can get the content. With the API, The Guardian tracks usage patterns on end users using the application powered by its API, including which news or topics are more popular among which devices, what time of the day or night and so on.
By capturing usage metrics, the API provider knows *what was requested and by whom?*, *what did the API delivered for the respected request?*, *what content was shown to the end user?* and *what did the end user did with the content that was shown?* While answering these questions, the API provider can make better and more accurate decisions for the target audience. [DW11]

11.1.1 Request and Response metric

Requests and response metric is used to understand how the API works. From this, the API provider understands what trends are emerging, what is being requested more and what is being delivered.
Although this is a useful metrics, they do not tell how this is being translated into a value for the end users. To know what end users are doing with the provided data through the API, the API provider needs to map this metric to data that is captured by client applications. [DW11]

11.1.2 Impressions

Impressions are metrics that capture actual consumption of the content that is being delivered. Impressions would be the number of times the provided content is displayed, no matter if the end user interacted or not.
For example, suppose you have 100 followers on Twitter, and you tweet something twice. The reach is 100 because the number of followers did not change. However, you have 200 impressions because every single follower would see both tweets.
As a result, the impression numbers could be higher or lower than the request and response data. This metric is relevant and useful for APIs that are driven by what is displayed not what is requested.
A way to capture this metric is by adding a callback function that returns the information to the API from the calling client. Also by placing an image beacon in the content. In this way the beacon signals from the server when the calling client presents it.[DW11]

11.2 Operational metrics

Unlike a website which can run JavaScript inside the browser to analyze the gathered data and to get more in-depth insights about its users, API analytics have to generate on the server. On the server, it is possible to insert tools at any layer which can determine the traffic through the API and what does it mean for the API provider. Rather than having system files with IP addresses, URIs and error codes, the API provider can create a database with this information. Furthermore, metrics using this information, improve the ability to better support the API consumers – developers and end users. When developers complain about the API functionalities, the API provider can capture the payload, debug it and identify the issue. [DW11]

11.2.1 Effectiveness metrics

If the metric is measuring how well the API is contributing to the desired result, it is related to effectiveness metric. Effectiveness is about achieving the goal or following the right path. Effectiveness metrics can help the API provider to realize how to make the API more useful for confirming the business strategies. An excellent example is a case that happened at Netflix. Netflix realizes that it was receiving around 30 billion requests per month and some of the interfaces that were powered by the Netflix API were complaining. The reason was that the API was not providing sufficient API calls, useful for the clients. This issue slowed down each client and resulted in more load in server systems. The raised problem was a good indication that the API was not serving its clients as efficiently as it could. Understanding the problem, Netflix improved the API in the following versions which led to improvements in performance and client satisfaction. [DW11]

11.2.2 Performance metrics

Metrics that track overall API performance include error rates, types of errors, system performance, latencies in request handling, and timeouts. Such metrics should be monitored over time. Additions to the API should not slow it down. These kinds of metrics often tie in directly to the Service Level Agreements and monitoring systems that are in place for real-time responses when thresholds are met. Particular attention should be paid to the impact of code and system changes, by monitoring the effects on internal servers as well as the experience on the developer side. [DW11]

Availability and uptime

One important metric when it comes to something as high demand as an API is availability and uptime. Uptime tells the API provider the amount of time in which the resource is technically unreachable, which means that describes whether the service is on or off. While on the other side, availability cares more about whether or not the access was full, unrestricted and unhampered. Availability tracks how often the API has failed until now and the reason why. Together these metrics must work together. Uptime is not everything, and without uptime, availability cannot be determined.[San17d]

Responsiveness and latency

Tracking responsiveness and latency add another layer to the concept of availability, by considering whether or not data was easily interactable and

was surely responsive to requests within a reasonable, specifically defined time set. [San17d]
For example, the API provider could have 99% uptime and 99% active API call service, but if each call takes 2 hours to respond, leads to a poor user experience and questionable usability.
Availability and uptime tell if there's a problem while responsiveness and latency are cued into identifying why problems exist.

Endpoint evaluation

Endpoint evaluation metrics take care of what is going on with the endpoints. In more details, this metric, answers the following questions:

1. **Frequency:** Does the endpoints that are used more often than others? At what rate are they used?
2. **Utilization:** What are the least used endpoints?
3. **Traffic:** What endpoints are hit the most with malicious traffic?
4. **Vulnerabilities:** When data breach does occur, or when penetration testing has shown vulnerability, what endpoints are responsible?

Answering these questions can inform API providers about the health of an API in general, including platform insights such as the following.

1. **Bloat:** If the API provider has a significant number of available endpoints, it suggests that the provider is supporting large portions of a codebase that is no longer needed.
2. **Service Efficiency:** If the API provider has one or two endpoints that are always hit, those should have a look. If these endpoints cover multiple services, these services should be broken into their endpoints for a correct microservice approach.
3. **Security:** If the API provider has vulnerable endpoints, it is an issue with the codebase allowing for vulnerabilities to be easily detected and exploited.
4. **Vulnerabilities:** When data breach does occur, or when penetration testing has shown vulnerability, what endpoints are responsible?

While issues might be at any step in the API system, identifying the symptoms at the endpoint level informs and helps the provider to solve the issues. [San17d]

11.3 Wrap up

To measure the success of the API, the provider should use a variety of metrics.
Usage metrics, are metrics that help the provider to better know how the API consumers are interacting with it, including, request and response metric, which tells the emerging trends, what is being requested and delivered and impressions metric, which capture the actual consumption of the delivered content.
Additionally, operational metrics, including effectiveness metric, which measures how well the API is contributing to the desired result, helping the provider to realize how to make the API more useful regarding the business strategies. Also, performance metrics that track the overall API performance, including error rates, error types, system performance, latencies and timeouts.

Part V

Retiring Phase

Chapter 12

Retiring

In this chapter, I will examine the retirement state of the API lifecycle in participating in the API Economy. In API lifecycle, retirement does not have a single meaning. An API version may be scheduled for depreciation or the API is still functional in limited formats with restricted access control. There are five common causes that will push APIs towards retirement.

12.1 Lack of third party developer innovation

Lack of third-party developer innovation is a scenario that may cause the retirement of the API, which means that the API is still receiving limited or unmeaningful use by developer or end-users. However, this could be due to poor marketing, or the API is not giving any value to the community. [Doe17]
A great example of this case is Netflix API. In 2009, Netflix released its API program with the reason for new third-party integrations by external developers. However, in 2014, Netflix stopped issuing new API tokens to developers and later on that year they shut it down for public use. The reason was that Netflix public API received for 5 years 0,3% of the total API traffic. In other words, 11 years of traffic in public API are equal to 1 day of private API traffic. [Jac14] From this case, we can learn that when a brand name and its native platforms are powerful, creating an ecosystem with the community is not necessary and will lead to retirement.

12.2 Opposing financial incentive, competition

An API may be retired if the API conflicts with the business goals of the company. While exposing data for anyone to consume, API providers run the risk of losing a market advantage, especially if this is taking business away from native distribution channels. [Doe17]
A great example is Strava. Strava offers fitness software with an API to extend their brand. Since the version 3 of the API, Strava restricted developers, giving them a limited access program. These new requirements ban "applications that encourage competition with Strava" as well those that "Replicate Strava functionality." A considerable API retirement can occur when developers are incorporating monetization techniques into third-party applications that break the original terms of the agreement and start a competition with the API provider. [Doe17]

12.3 Changes in technology & consolidating internal services

Technology is continually changing, evolving and these evolvements have the potential to make APIs out-of-date. Often happens during internal redesigns, acquisitions, or external industry advancements. API retirement could arise from evolving protocol trends, such as GraphQL replacing REST, or REST replacing SOAP, or changes in end-user experiences, such as Netflix changing its business model from DVD distribution to media streaming, or Fedex which updated its APIs to Fedex Web Services, replacing XML based tracking applications with a large update. [Doe17]

12.4 Versioning

Versioning is a common practice to retire an API and can arise when significant alternations to the API are required. Any evolving API will require versioning sooner or later. When and how versioning will occur is based on the expectations of the API consumers. [Doe17]
Even though versioning strategy may vary based on the API consumers, I recommend an API versioning strategy based the information from the literature research. When an API is is a pre-release, the API provider should gain some feedback from developers, to establish expectations that the API could change. At this phase, the API will remain at version 1 for a period.

The API will be volatile for the consumers, and they should expect that. Once the API provider decided to release the API, it will be as a contract between the provider and consumer that the API cannot contain changes that break the application, powered by it, without new version releases. API versions are major, minor and non-breaking. Breaking changes occur in a major version. Developers must manually migrate to this major version and update the application to confirm the changes. While for minor and non-breaking changes, developers are automatically migrated to the latest version.
Two great examples about versioning are the case of Youtube and Twitter. When Youtube released version 3 of the API, it surprised the developers when they found that it was not possible to create Youtube applications for SmartTVs. [Doe17]
Furthermore, Twitter API version 1 was retired in 2013. The new version update added OAuth. This means that performing the necessary act of reading a public timeline will, first of all, require an application to be created with Twitter, that application is configured with a secret client key, and the application is completing the OAuth handshake before using the API. Twitter did this to enforce call rate limiting with less liberal access, reducing the system load, decreasing API vulnerability and promoting their embeddable tweet program. [Mom13]

12.5 Security issues

API providers may choose to discontinue the public API because of security concerns associated with making internal data public. Like the example of Twitter which added OAuth in the new version of the API to decrease vulnerability issues, Google Earth API was discontinued because it relied on outdated technology like NPAPI framework, which had become a security concern for Google. [Doe17]

12.6 Preparations for retirement

A software ecosystem is a collection of software systems which are developed and co-evolve in the same environment. An ecosystem like this which can be powered by APIs can be easily destroyed when an API is deprecated. [Lun12] Developers are depended on the service which the API provides, to support their products. A complete deprecation can cause adverse reactions and lower the company's reputation if the retirement is not managed well. To keep an excellent reputation in the community, it is recommended for the

API provider to be transparent using the following techniques to prepare the API for retirement or complete shutdown.

12.6.1 Schedule a retirement plan

The API provider should announce at least six months in advance about a deprecation. This announcement will hopefully give developers enough time for new integrations or partnerships. If individual teams of developers are struggling to adopt the change, the API provider should consider extending the time frame. Google usually for Maps and Youtube API follow one year in advance to announce. Sadly, Twitter uses only three months. [Doe17]

12.6.2 Make changes in stages

If the API is still operating, shut down earlier, older versions that receive limited use. [Doe17]

12.6.3 Public announcements

Announce explicitly into a blog post the API changes. Outline when the changes will go into effect, who will be affected and any other information that the community needs to know. Moreover, the announcement should be promoted into different channels to reach the target audience. [Doe17]

12.6.4 Ease the transition

If the API has been versioned or merged into another service or API, explain the process of transition. If the transition process is not transparent, the developer can have some hard time to transit. [Doe17]

12.6.5 Blackout testing

A blackout test can be defined as a planned, timebox event when the API provider will turn off a particular API to help developers better understand the implications of the eventual retirement. Hopefully, a blackout test will help developers to have a clear idea of how the retirement will affect their applications. Youtube and Twitter have used blackout testing in the past. [And14]

12.6.6 Make sure not to validate the Terms of Service

Before retiring, the Terms of Service should be reviewed to make sure that no binding contracts are violated between partners or the provider and consumers. A way is to plan to write a deprecation policy whenever a new version of the API is created. [Doe17]

12.6.7 Include a deprecation message in the Sunset HTTP response

The Sunset HTTP response header field allows a server to communicate the fact that a resource is expected to become unresponsive at a specific point in time. It provides information for clients which they can use to control their usage of the resource. The Sunset header contains a single timestamp which advertises the point in time when the resource is expected to become unresponsive. [Wil16] While applying this, developers will expect that a particular resource will become obsolete in the near future. [Doe17]

12.7 Wrap up

Retirement can come from many reasons. Lack of third-party integrations, which means that the API is receiving unmeaningful traffic from third-party integrations. Also from competitions, which means the API conflict with the business goals of the company. Additionally from changes in technology which can make the API out-of-date. Moreover from versioning, which is a common practice to retire when a significant alternations are required, and also from security issues, associated with making internal data public.
Even though some developers quickly adopt the changes, some others may be unaware of API changes. They will not respond quickly to update their applications to be compatible with new changes. That is why is essential to prepare the developers for deprecation or retirement before actually doing it. Planning the retirement of an API may be a significant advance toward restructuring how the developers will receive the data.

Chapter 13

Conclusion

13.1 Wrap-up of Findings

In this section, I want to condense the information regarding to the research questions, in which my thesis is build upon. I wrap-up the findings separately.

RQ1: What is API and API Economy?

API stands for application programming interface and is a machine-readable interface through which software applications make their functionality or data accessible to another authorized application. API exposes some of the application's internal functions to the outside world through a defined interface. In this way, the outside world can use the functionality without knowing how the internals of the application work. The goal of APIs is to ease and accelerate the development of software applications by providing a part of its functionality to the outside world, so developers do not lose time implementing the solution themselves.
APIs being part of products, are part of an agile business methodology, helping companies to create new revenue streams, by selling data on demand or by attracting investors, partners, producing an economy around itself, called **API Economy**. The API Economy is the economy where companies expose their internal business assets or services in the form of APIs to third parties, which ca be partners or external developers, with the goal of unlocking additional business value, accelerating loyalty, and customer growth through the creation of new asset classes.

RQ2: Which are the lifecycle phases of an API participating in the API Economy?

The answers regarding this research question are combined with the following subsection.

RQ3: What are the components of each API lifecycle phase?

Based on systematic literature review, manual searching in specific conferences, papers, journals and highly cited blog posts, like ACM Digital library, DBLD, IEE Xplore, Google Schoolar, Nordic API, API Evangelist and books related to Web APIs, I found out the lifecycle phases of APIs participating in the API Economy. The lifecycle phases are not explicitly defined in the information sources, rather than my perception based on these sources of information.

Starting with the **planning phase**, in which the company has to see the API as a product that needs to be develop with some business strategies in mind, and to move the business toward its goal, together with some legal considerations in mind for distributing the API data. I introduced and analyzed the business strategies: **private**, **partner** and **public** models, together with their business benefits, disadvantages and challenges. Even though private APIs are not considered part of the API Economy based on the API Economy definition, it is an excellent strategy to start with a private API and then transit to partner or public. Additionally, I introduced nine monetization models that are categorized in three groups: **developer pays**, **developer gets payed** and **indirect**. These are essential to consider how the API will return the investment and boost the revenue. Moreover, I introduced the legal aspects that should be considered when planning to expose the business assets and data through the API. Not understanding the ownership of the content that an API is providing, can bring legal implications for the company.

Continuing with the **design phase**, in which the company should think about the API design. The design of an API is important and not easy. A good API design improves the usability of it, resulting in higher adaption, higher user satisfaction and an overall better chance of success. While on the other side, a bad user experience while using the API may lead to a bad reputation.
I introduced and analyzed different usability heuristics that can be implemented based on Nielsen's 10 heuristics, giving a direction of how to craft an

API.

Furthermore, with the **build phase**, in which the company should develop and bring the API into life while deploying it. I introduced and shortly analyzed the methodologies and tools that can be used to develop and deploy an API.

Moreover, with the **operational phase**, in which the company thinks how to market, engage developers and measure the success of the API using a variety of metrics. I introduced and analyzed marketing and engagement strategies to obtain best possible results and also a variety of metrics, like request and response metrics, impressions, effectiveness and performance metrics, to measure the growth and success of the API.

Lastly, the **retirement phase**, in which the company decides to retire the API. There are many reasons for API to retire. Lack of third-party integrations, competition, changes in technology, versioning and security issues. Furthermore, to keep an excellent reputation in the community, some preparations for retirement are necessary, because in an ecosystem like this, which is powered by APIs, can be easily destroyed when an API is deprecated. Retirement can cause adverse reactions and lower the company's reputation if it is not managed well.

13.2 Limitations & Future work

This master thesis presents an API lifecycle analysis for participating in the API Economy network. The analysis of each lifecycle phase, together with its components, is supposed to help companies in participating and succeeding in the API Economy.
However, this master thesis is based only on the manual search for information using systematic literature review approach. Due to a lack of detailed information, besides the developer documentation, from APIs that already thrive in the API Economy, it is not possible to compare my findings of API lifecycle with theirs. More detailed information from companies or enterprises regarding how they manage the lifecycle of the API would have helped to improve the analysis of the API lifecycle, making it more accurate. Furthermore, in each lifecycle phase, I have used examples only from publicly available APIs, because information regarding APIs under a partnership strategy is not disclosed and could not be found.

Bibliography

[Ama18] Amazon. What is devops, 2018.

[And14] Callum Anderson. Mendeley api – blackout testing, 2014.

[API15] Nordic APIs. *Developing the API Mindset. Preparing Your Business for Private, Partner, and Public APIs.* 2015.

[API16] Nordic APIs. *API Driven DevOps.* 2016.

[BK09] David Budgen Mark Turner John Bailey Stephen Linkman Barbara Kitchenham, O. Pearl Brereton. Systematic literature reviews in software engineering – a systematic literature review. *ELSEVIER*, 2009.

[Blo] Jason Bloomberg. Architecting the composable enterprise. *MuleSoft.*

[Cor16] IBM Corporation. Innovation in the api economy. building winning experiences and new capabilities to compete. *IBM Institute for Business Value*, 2016.

[Doe15] Bill Doerrfeld. Api lifecycle analysis stage: Preparing your prelaunch api strategy, 2015.

[Doe17] Bill Doerrfeld. Api lifecycle retirement stage: A history of major public api retirements, 2017.

[Doe18] Chase Doelling. Scalable apis are built from consistency, 2018.

[DW11] Daniel Jacobson Dan Woods, Greg Brail. *APIs: A Strategy Guide.* O'Reilly Media, Inc., 2011.

[Goo13] Google. Google, a fresh new look for the maps api, for all one million sites, 2013.

[Hal] Mahesh Haldar. Api designing guidelines.

[Jac14] Daniel Jacobson. Top 10 lessons learned from the netflix api - oscon 2014, 2014.

[Jen15] Claus T. Jensen. *API for Dummies.* John Wiley & Sons, Inc., 2015.

[Kep14] Ben Kepes. Software may be eating the world but apis are giving it teeth. *Diversity Limited*, 2014.

[KH14] Dr. Ali Arsanjani Kerrie Holley, Samuel Antoun. The power of the api economy. stimulate innovation, increase productivity, develop new channels, and reach new markets. *IBM Institute for Business Value*, 2014.

[Lan12] Kin Lane. The secret to amazons success internal apis, 2012.

[Lan14] Kin Lane. Taking a quick look at the leading api partner programs, 2014.

[Lan16] Kin Lane. History of apis, 2016.

[Lau] Page Laubheimer. Preventing user errors: Avoiding unconscious slips.

[Lun12] Mircea Lungu. How do developers react to api deprecation? the case of a smalltalk ecosystem, 2012.

[Mah14] Michael Mahemoff. Don't api all the things ... the downside of public apis, 2014.

[Mai11] MailChimp. 10m+ api calls per day and more, 2011.

[MB13] Dinesh Venkateswaran Manfred Bortenschlager, Graham Thomas. Leveraging apis as part of digital strategy. *Wired*, 2013.

[McK16] McKinsey. The social economy: Unlocking value and productivity through social technologies. Technical report, 2016.

[mic16] microsoft. Api design, 2016.

[Mit] Ronnie Mitra. Improve your api design with 7 guiding principles.

[Mom13] Matthew Mombrea. The death of the twitter api or a new beginning?, 2013.

[Mus13] John Musser. Api business models, 2013.

[Net11] Netflix. Redesigning the netflix api, 2011.

[Nie95] Jakob Nielsen. 10 usability heuristics for user interface design, 1995.

[Ped14] Bruno Pedro. How to release a free api and get paid indirectly, 2014.

[PM16] Mark O'Neill Paolo Malinverno. Magic quadrant for full life cycle api management. *Gartner*, 2016.

[Pro11] ProgrammableWeb. Api growth doubles in 2010, social and mobile are trends, 2011.

[Pro13a] ProgrammableWeb. Programmable web's directory hits 10,000 apis. and counting, 2013.

[Pro13b] ProgrammableWeb. Programmableweb api directory eclipses 17,000 as api economy continues surge, 2013.

[Pro14] ProgrammableWeb. Private, partner or public: Which api strategy is best for business?, 2014.

[Rig15] Jennifer Riggins. Top 5 api monetization models, 2015.

[San16a] Kristopher Sandoval. Living in the cloud stack – understanding saas, paas, and iaas apis, 2016.

[San16b] Kristopher Sandoval. What data formats should my api support?, 2016.

[San17a] Kristopher Sandoval. Best practices for api error handling, 2017.

[San17b] Kristopher Sandoval. The end of api documentation as we know it, 2017.

[San17c] Kristopher Sandoval. Is graphql the end of rest style apis?, 2017.

[San17d] Kristopher Sandoval. Using api analytics to empower the platform, 2017.

[San18] Kristopher Sandoval. 7 items no api documentation can live without, 2018.

[Sma16] Smartbear. A global survey looking at the growth, opportunities, challenges and processes for the api industry in 2016. Technical report, 2016.

[Sta17] Stackify. Ci tools, 2017.

[Stu17] Sashko Stubailo. Graphql vs. rest, 2017.

[Tea16] Github Engineering Team. The github graphql api, 2016.

[Who] WhoIsHostingThis. Http status codes: The complete list.

[Wil16] E. Wilde. The sunset http header. *IETF*, 2016.