Know Your APIs
A Guidebook For Beginners

edited by

Hussain Fakhruddin

Teks mobile
TEKNOWLEDGE SOFTWARE

Visit us on LinkedIn at:
https://www.linkedin.com/company/teksmobile

About Teksmobile

Teksmobile is one of the fastest growing mobile app development companies in the world. We specialize in the development of custom applications for the Apple (iOS, watchOS, tvOS, macOS) and Android (smartphone, tablet, wearables) platforms. Team Teks was formed in early-2006, and to date, we have successfully created more than 1000 mobile apps.

Know Your APIs: A Guidebook For Beginners

Edited by: Hussain Fakhruddin (CEO, Teksmobile)

'***Know Your APIs: A Guidebook For Beginners***' is an exhaustive technical ebook brought to you by Teksmobile, purely for learning and instructional purpose. Please use it as such!

Contents

Preface

The API economy is big, and it is growing bigger all the time. Right now, APIs are the heart and soul for the next-gen digital economy, facilitating seamless, powerful backend cloud connectivity to web and mobile applications, driving elements in the IoT domain, and in general, enabling developers to do more with existing resources. It's not for nothing that APIs have been dubbed as the 'fossil fuel for the next generation' and 2017 been named as the 'Year Of The API Economy'.

In the ebook you are just about to start reading, we have outlined all the important points related to APIs. This book is meant for newbies in this field - and for their convenience - we have started out with the very basics, and have then moved on to API design principles, lifecycle analysis, API tools and best practices, and the usage of APIs in different industries. The book rounds off with 6 fascinating API case studies.

We trust that you will find the topics covered in this book to be extensive and beneficial.

Happy Reading!

-- Team Teksmobile

1. The Basics

In this chapter, you will learn:

- *The meaning and significance of APIs.*
- *The reasons for implementing APIs.*
- *A brief idea about how APIs are deployed.*

1.1. What Are APIs?

APIs, or Application Program Interfaces, are tools that make big data easier to work with. In essence, APIs serve as viable platforms for multiple pieces of compatible software to 'interact' with each other. The next time you find Facebook displaying your Instagram images, or Uber quickly tracking your location on Maps - you should know that APIs are pulling the strings.

In the context of mobile application development, APIs are the tools/protocols/standards that help apps connect with the cloud in the backend (backend-as-a-service, or BaaS). They offer

seamless programmatic access to proprietary applications for developers, doing away with the need for sharing codes repeatedly. With APIs, developers can build new capabilities, integrate customized third-party services, create new models or make modifications in existing ones, and add greater value to their products.

Considering the many advantages that APIs bring to the table, the recent surge in their popularity has not been surprising. In February 2017, the total number APIs on ProgrammableWeb was 17007[1] - more than double the figure recorded in November 2012. In fact, this year has been touted as the 'Year Of The API Economy' by experts across the globe.

1.2. Why Are APIs Used?

As business platforms get increasingly advanced and digitized, the importance of smooth-functioning APIs continues to soar. These custom interfaces allow entrepreneurs to leverage their businesses in the most optimal manner, by implementing agile standards, expansion of corporate branding, and shortening of the time-to-market spans.

[1] → *as on 17/3/2017*

The best thing about APIs is that, business owners do not need to have prior programming knowledge to use them. There are plenty of reliable API providers with whom they can partner, to open up new market channels, connect with new partners, and drive their businesses forward. With APIs, innovation also becomes easier, and significantly quicker.

1.3. How APIs Are Used

Developers use APIs extensively to enhance the functionalities of their mobile applications. However, the adoption of these platforms is far from being limited to this. General users (i.e., non-programmers) regularly come across common interfaces like the Facebook API, the Twitter API or the Google Maps API. In addition to the mobile platform, APIs are also used on the Web and even in apps for wearables.

Since the first instance of the use of APIs (by Salesforce in February 2000), the interfaces have come a long way. At present, APIs find widespread usage in practically every industry - right from sports and music, to social networking, payments & finance, and travel. Just as websites were at the turn of the decade, APIs are also moving on from

being a ‘nice option’ to ‘must-have tools’ for business owners.

We will close out this section with a very basic idea of the components in an ‘integrated’ API framework/system. On the one side, there are the systems on which the APIs exist - the ones which ‘provide’ these interfaces. These are called the ‘servers’. Receiving the APIs are the ‘client systems’ - programs that can customize APIs to make them best suited for the requirements on hand. Finally, there are the ‘users’ - the final customers who specify how a ‘client system’ is going to interact with an API.

---

2. API Designing & Management

In this chapter, you will learn:

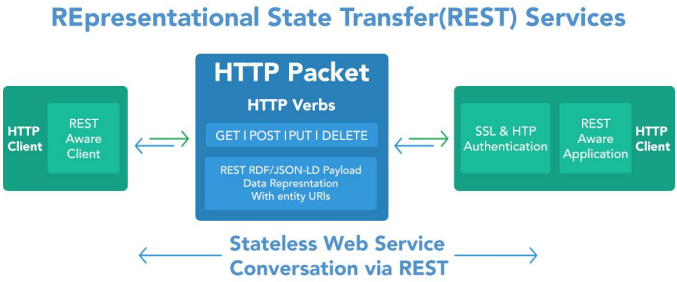
- *Key factors related to API designing*
- *API request methods and HTTP status codes*
- *API design best practices*
- *API data formats (JSON/XML)*
- *The API lifecycle*

APIs are already big, and growing bigger. By the end of 2017, custom APIs will boost the total economic value of mobile applications beyond the $230 billion mark. In the domain of product marketing too, APIs have started to leave a mark (the case study of Expedia is a classic example). It has become of utmost importance for API providers to be aware of the best practices of designing high-powered, user-friendly interfaces. We turn our attentions to such design-related factors in this chapter.

2.1. Protocols

In the context of API architecture, ‘protocols’ refer to the standards or ‘etiquettes’ which govern how processes are monitored and handled by the software. Just as HTTP is the main protocol for the World Wide Web, REST, SOAP and XML-RPC are popular protocols on the basis of which APIs are designed. Let’s examine them in more detail.

2.1.1. REST

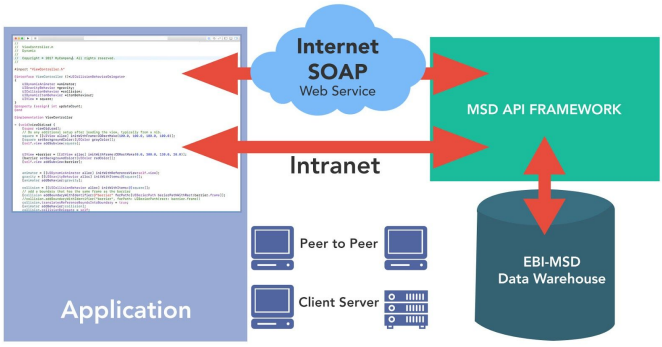


REST, or REpresentational State Transfer, makes use of standard HTTP architecture and advanced XML documents for the implementation of secure, high-quality web services. There are 4 ‘methods’ (to be discussed later) in these APIs, through which REST interacts with the server and prompts the desired response. The ‘Uniform Resource

Identifiers' (URIs) have to be carefully defined, for the creation of this type of APIs. Typically, REST is faster and more 'lightweight' than SOAP - since it uses up less resources.

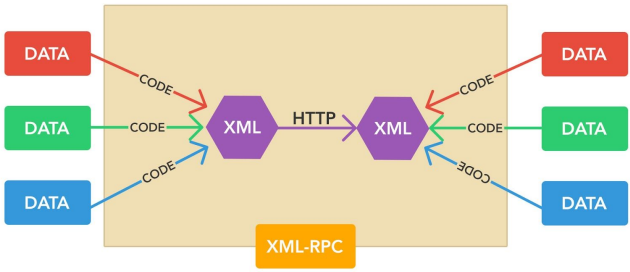
2.1.2. SOAP

SOAP, or Simple Object Access Protocol, is another widely-used design standard for APIs. It is exclusively based on XML, and facilitate communication between distributed/shared apps - often located on non-similar platforms (ie., the protocol is platform-independent). SOAP puts relatively higher load on networks (the requirement to put tags is an important factor) in comparison to REST. In most cases, external libraries are required to parse SOAP responses. The intensive serialization required in the SOAP architecture places contributes in making it 'heavier'. In contrast, REST is 'stateless'.



2.1.3. XML-RPC

RPC, or Remote Procedure Calling protocol, transforms discrete XML-RPC messages into easy-to-understand HTTP-POST requests. The API responses are recorded in the server, and the responses are provided in the XML format.



REST APIs, with more than 70% of developer mindshare, is, by far the most popular API designing protocol. SOAP, with a 20%-odd share, is a distant second, while XML-RPC has a sub-10% share.

Note: Since RESTful techniques making up for the majority of public APIs, we will primarily be concerned with it in our discussions.

2.2. Request Methods

To ensure correct API performance, the ‘server’ needs to ‘understand’ the type of action(s) that the ‘client’ wants it to take. This information is passed on in REST APIs with the help of 4 separate request methods.

Actions Handled By Resource Controller

Verb	Path	Action	Route Name
GET	/image	index	image.index
GET	/image/create	create	image.create
POST	/image	store	image.show
GET	/image/{image}	show	image.index
GET	/image/{image}/edit	edit	image.edit
PUT/PATCH	/image/{image}	Update	image.upadate
DELETE	/image/{image}	destroy	image.destroy
GET	/image	index	image.index

2.2.1. POST

The POST request method ‘creates’ new subordinate resources, under existing parent resources. It is typically used for non-idempotent requests.

2.2.2. GET

The GET verb has to be used to retrieve existing resources. Developers, particularly the new ones, often make the mistake of using GET to make modifications to resources. That cannot be done with this request.

2.2.3. PUT

This is the request method used in RESTful architecture to update or edit an existing resource. In addition, PUT can be used to create new resources - only in cases where the client selects the ‘Resource ID’ (instead of the server).

2.2.4. DELETE

DELETE is an idempotent request method, used to remove an existing resource. A resource has to be identified by the Uniform Resource Identifier (URI), prior to the deletion. Typically, if a resource has already been deleted, calling DELETE on it again returns a ‘NOT FOUND’ response.

Apart from these 4, **PATCH** is yet another often-used RESTful request method. It is used to modify resources, and generally specifies the process in which these modifications are done as well.

2.3. HTTP Status Codes

On receiving a request from the 'client', the 'server' makes an attempt to process it. Based on how that goes, a HTTP status code response is generated - in the form of a three-digit number. We will take a tour through the most common HTTP status codes over here.

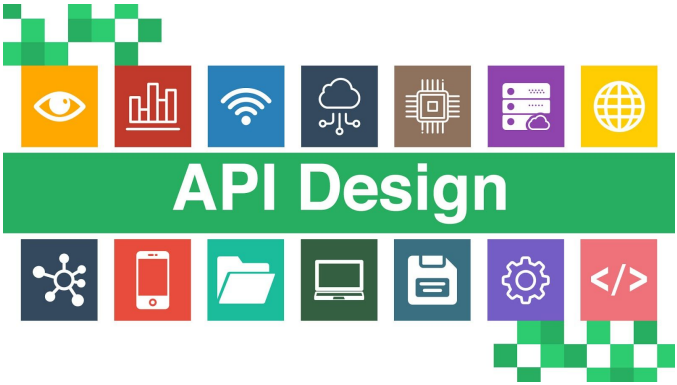
#200 - OK
#201 - Created
#204 - Zero Content
#301 - Moved Permanently
#304 - Not Modified
#401 - Unauthorized
#404 - Not Found
#422 - Unprocessable Entity
#500 - Internal Server Error

The Following are The list of HTTP response status codes

Type	Status Codes	Examples
Informational	1xx	100 Continue, 101 Switching Protocols
Success	2xx	200 - OK , 201 - Created, 202 Accepted
Redirection	3xx	300 Multiple Choices, 301 Moved Permanently, 302 - Found
Client Error	4xx	404 Bad Request, 403 - Forbidden, 404 - Not Found , 422 - unprocessable Entity
Server Error	5xx	500 - Internal Server Error , 503 - Service Unavailable

Classifying broadly, all 1xx HTTP status codes are informational in nature, the 2xx ones indicate different forms of success, 3xx codes are all about redirection, client-errors are indicated by 4xx responses, while 5xx responses present server-side errors.

2.4. API Designing: Best Practices



Just as software applications need to deliver top-class user-end experience (UX), APIs too have to be smartly designed to ensure satisfaction of the customers. More importantly, unless an API is designed in accordance with the best practices, it might not deliver all the desired results, and using it can prove to be a hassle for developers. We will focus briefly on how RESTful APIs should be designed next.

2.4.1. Versioning Is Important

Over time, APIs have to be updated. The last thing an API provider wants is a new version affecting the final performance of the interface, altering the customer-experience. By adding versions in the URL of an API, two purposes are served: firstly, new versions, with additional features and functionalities, can be released without modifying the interface per se, and secondly, the user-distribution of an API can be systematically tracked.

2.4.2. Data Caching Should Be Included

While in-memory caching for every host makes an API heavier and often buggy, developers typically include cloud-based hosted caching solutions - to bolster both the speed and efficiency of interfaces. A properly set up distributed caching service allows hosts to alter/modify APIs without having any operational effect on the latter.

2.4.3. Hypermedia For Scalability

Over time, business APIs are expected to do more. The digitization scenario is evolving rapidly, and API providers have to make their tools seamlessly scalable, to be able to meet up the increased requirements. Hypermedia, which refers to hypertext extensions for multimedia functionality,

help in ensuring that client-side apps do not get damaged in any way, as the APIs are changed and scaled up. The role of hypermedia as the ideal tool for continuous API evolvability was touched upon at the 2016 Nordic API World Tour.

2.4.4. Pagination Matters

Pagination, when done consistently, improves the efficacy of API tools in two important ways. First, they reduce the total load of computation on the app servers. From the perspective of clients, non-important data is stopped from being transferred. This, in turn, enhances the overall usability of APIs significantly. In addition, annotations and markers are vital for high-level authentication and access control.

2.4.5. A Simple API Is A Good API

The more complex an API is, the greater are the chances for it to go wrong somewhere. Including too much of rich data can make the development cycle of APIs longer, while having multiple structural variations is not a good idea either. The core features of an interface have to be identified and closely monitored - right from the data sequencing and workflows, to the built-in software options and functioning data structure. Unless software developers find an API to be

easy-to-understand and easy-to-use, it won't be of any value.

The security parameters of APIs also have to be carefully set up (many APIs are simultaneously available on HTTP and HTTPs). To avoid overloads and API concurrency issues, certain rate limits should be pre-specified. As already highlighted earlier, the HTTP response codes of REST APIs have to be correct as well.

Like any software application, APIs also have to be carefully tested before release. There are several popular tools for API testing, and we will highlight a few of them in a later section of this ebook.

2.5. Data Formats

Recall the definition of APIs we started with: they help software tools/applications to 'interact' with each other. For this, the 'language' of APIs needs to be in a data format that is machine-understandable, and easily usable by humans (in this case, coders). Here's a brief look at the two most popular data formats for APIs:

2.5.1. XML

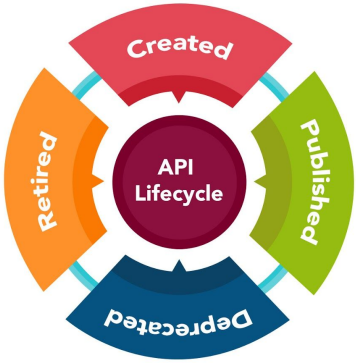
The older of the two data formats (used since 1996), and also the ‘heavier’. XML comes with relatively simple API data structure support, with the help of user-friendly building blocks. Primary blocks in the XML architecture are called ‘nodes’ - and the format typically starts with a root/parent node and then moves on to the ‘child nodes’ within. The value contained within the opening and closing tags of nodes specifies the details about a component. In comparison with JSON (which we will discuss next), XML is a considerably more ‘verbose’ data format.

2.5.2. JSON

With much lesser markups and necessitating low volumes of data transfer, JSON is the ‘faster’ alternative to XML as a workable data format for APIs. One of the reasons behind its popularity is the fact that JSON is wholly based on Javascript - and can be used for web applications in both the backend and the frontend. The format uses ‘keys’ (that specify attributes) and ‘values’ (that contain the details) for communicating with supported systems. JSON also uses building blocks, but the format is relatively simpler and ‘lightweight’ than XML (no opening/closing tags required either).

The data formats used in APIs have to be mentioned under ‘Content-type’.

2.6. API Lifecycle Management



All types of software-as-a-service (SaaS) have their own distinct lifecycles. APIs are no exception to this rule either. In a sentence, the API lifecycle refers to the entire span of time from the conceptualization of the tool, right up to its maturity, deprecation and final retirement. There are no static rules about the exact duration of the API lifecycle (or indeed, of the different stages) - which varies with the precise nature and purposes of each interface. In this section, we will present the 4 broad stages that the lifecycle of APIs are divided under.

2.6.1. Stage 1 - API Requirement Definition

During the first, or the planning stage, all the important details about an API are sorted out. These details must necessarily include the justification of creating the tool (an API must not be created just for the heck of it), its primary objectives, and the outlines of the business policy/development policy within which it will be implemented.

The basic API blueprint is laid out during this stage. The blueprint needs to shed light on each of the following points:

- API growth model
- API usage projections/predictions
- API mission statement
- Expected returns
- API marketing/promotional methods
- Data format (JSON or XML) and protocol (REST or SOAP)
- Implementation of API-first approach.

As is evident, the background research about designing an API has to be done in the planning stage. Of course, providers also have to take the call of whether they would create a public API, a private API, or a partner API.

2.6.2. Stage 2 - API Development & Integration

Following the planning phase, the attention next shifts to API development - and the coding required for the purpose. There are several user-friendly development tools for interfaces out in the wild (we will name a few in the next section) - that go a long way in aiding app developers, and shortening the duration of this phase, as a result. API providers typically chalk out the versioning plans, the scalability options and the size of their interfaces by this stage. The following factors have to be placed importance on during the development stage:

- API designing, for user-understandability and machine-understanding
- API construction, involving the main coding
- API authentication, validation and security
- API testing

By the end of the development phase, the API has a properly functioning two-way connection with the software application it has been integrated into. Advanced, multi-featured API development frameworks are available, which should be used at this stage. If required, developers can go back to Stage 1 too (the API lifecycle stages are iterative).

2.6.3. Stage 3 - API Operations & Management

A custom API management layer is created and initiated during the third stage of the API lifecycle. The focus of this layer is on the upgradation of 'API intelligence', and the resultant thrust in its efficiency/performance levels. The management layer performs this function by closely monitoring API analytics and stats, making user-behaviour more predictable, securing the 'endpoints' of the interface, and setting up a robust API monetization scheme.

This stage also requires coders to maintain and add to the API documentation (started from the development stage). The documentation eases the hassles of making changes and adding new features in a big way. The release notes and/or the changelog should include all the changes made in the API structure - right from the time it was conceptualized.

A well-designed API management layer has certain must-have features. These include:

- Access controls and security parameters
- Developer portal access
- API contract(s)
- Usage and analytics information

- ❖ Sandboxing (for test integration)
- ❖ Billing information

After the establishment of the API management layer, developers have to work with 2 separate URLs. The first one resides between the management tab and the client application/website. The second URL is the one between the API and the management tab.

Note: In order to make changes in the technical features or development aspects of the API, coders often have to move back to the previous stage, and return.

2.6.4. Stage 4 - API Engagement, Adoption, Maturity & Retirement

Once the API has been carefully tested and found to be free, it is ready to release - and the focus shifts on maximizing its 'discoverability' among web software/mobile app developers. To ensure glitch-free developer-experience (DX) and a high API value proposition, custom use-cases - highlighting the different important features of the interface - are also created. The main target is to drive up the overall engagement level of the API.

In the context of API engagement monitoring and expansion, API providers often find it useful to

create well-rounded ‘Developer Programs’. These programs are, in essence, comprehensive plans to increase the adoption rates of the software tool under question. The main considerations of an API Developer Program include:

- The developer portal (serving as the point of entrance).
- API evangelists (for promotions, both online and offline).
- Community development (for publicizing the API on social media).
- Pilot partner interaction (for feedback/opinions on API)
- Outreach acceleration (collaborating with partners to expand the reach of API-related objects).
- API monitoring (to track effectiveness of API at every stage).

As an API moves beyond its maturity stage, the returns from it starts to diminish. It has to be ‘retired’ (i.e., discontinued) when it becomes evident that it has entirely exhausted its value proposition.

The main reasons for retiring an API are:

- Lack of sync with business objectives.
- Slowdown (or complete stop) of innovation by app developers (API users).

- ❖ Mismatch of revenue objectives.
- ❖ Decline in user-base, indicated by metrics.
- ❖ New, unsolved, security threats.

Note: Along its entire lifespan, an API has to be managed with smartly coordinated API governance regulations. In the next chapter, we will take a look at API strategy optimization - which is extremely important for getting the full value out of APIs.

---

3. API Tools: An Overview

In this chapter, you will learn:

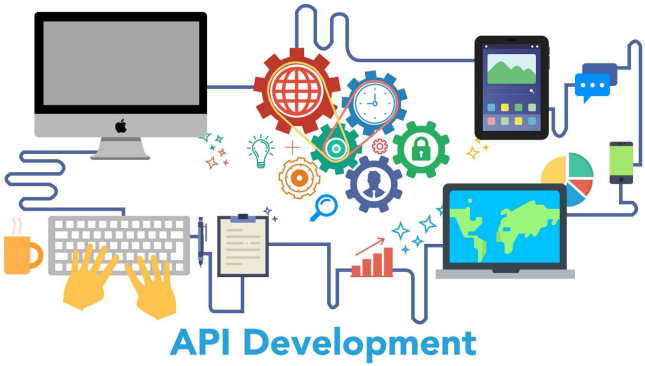
- A summary of the benefits of APIs
- API development tools, with particular emphasis on Swagger
- API discovery tools
- API testing types and tools
- The APIGEE platform
- API authorization tools
- API best practices (Designing, Security, Enterprise API handling, API strategy optimization)

Before taking you through lists of high-end API tools and software, let us quickly summarize the benefits of moving on to an optimized API platform. Working with APIs offers advantages at every stage, right through its lifecycle.

3.1. How Do APIs Help?

- App developers get to avail smooth self-service onboarding options.
- Agile development methods can be adopted, bringing down the overall development cycle.
- From the backend, applications can be run to scale.
- Cloud access becomes more reliable with software-as-a-service (SaaS).
- Avoiding software security threats becomes easier.
- Final users receive seamless connected experiences.
- Business reach grows with new partners and developers.
- Custom microservices for agile operations.

3.2. API Development: Tools



Coders can use different programming languages to create application program interfaces. Depending on their language preferences, there are several high-performance and user-friendly API development frameworks available:

- For Python → Django, Flask Web
- For Node.js → Express, Restify, Loopback
- For Ruby → Grape, Rails, Sinatra
- For PHP → Slim
- For Java → Spring, JAX-RS, Rest.li
- ASP.net
- AWS API Gateway

For API prototyping, tools like API Blueprint, AppNow, Apiary and Justinmind are popular among developers worldwide.

3.3. The Swagger Factor

Swagger, a widely used API description and documentation tool, was released in August 2011. Over the years, it has grown to become one of the most trusted API frameworks, with developers from around the world using it to create fully-customized, powerful interfaces. Swagger 2.0 has several useful open-source tools (available on Github) to aid in the designing, build stages and documentation of APIs.

3.3.1. Swagger Tools

3.3.1.1. Swagger Editor

The Swagger Editor allows coders to modify existing APIs as well as create new ones from scratch. The underlying Swagger definition is visually rendered on the editor. In addition, real-time feedback is also generated by the Swagger Editor. It uses YAML to author Swagger definitions.

3.3.1.2. Swagger UI

The Swagger UI simplifies the entire task of visual interaction with the platform by API developers. All components of the API documentation (as present in the Swagger specification) are present in Swagger UI. It is based on HTML5.

3.3.1.3. Swagger Codegen

The Swagger Codegen tool allows coders to generate client SDKs as well as smart stubs, right from the Swagger specification. The tools facilitates creation and usage of Swagger APIs in a wide range of popular programming languages.

3.3.1.4. Swagger Node

The Swagger Node tool is for Node.js users. It lets fully fledged design-driven server implementation of Swagger APIs.

3.3.1.5. Swagger Core

The Swagger Core libraries are extremely useful support resources for API development/documentation. These libraries are Java-supported - and help in the creation, consumption and general interoperability with Swagger definitions.

3.3.1.6. Swagger Parser

Swagger Parser lets coders parse built-in Swagger definitions from Java. It is a powerful standalone library used widely for definition parsing.

3.3.1.7. Swagger JS

This one is also a well-rounded Javascript library within the Swagger framework. From Node.js apps as well as from browsers, Swagger JS establishes connections with Swagger-defined APIs.

3.3.2. More From Swagger

Many API-related tools have successfully incorporated the Swagger framework to deliver services to API-makers. We have highlighted some of these Swagger-powered tools here:

Axway → offers interactive documentation for quick browsing and accurately testing available APIs.

Swagger Hub → free software-as-a-service tool for seamless coordination and collaboration right through the API lifecycle.

Restlet Studio → for visual presentation for web APIs and automatic generation of Swagger definitions.

Runscope → imports Swagger definitions and generates custom tests based on the defined methods.

APISpark → for smooth hosting, management, documentation and even testing of APIs.

Dreamfactory → used for both SQL and NoSQL for automatic generation of RESTful APIs.

RepreZen API Studio → an integrated environment for Swagger APIs. Helps in API designing, visualization, documentation, code generation and testing.

Gelato → a multi-featured API developer portal that is used to generate API reference documentation.

With built-in support for more than 25 languages, a base of over 2500 contributors and over 10 million downloads, Swagger easily counts among the most opted-for API frameworks available at present.

3.4. API Discovery: Tools

For web/mobile app developers, finding the 'right APIs' for their applications is not the easiest task in the world. The total count of APIs freely available on online repositories is increasing rapidly, and many new interfaces are released and added every week. In such a scenario, developers have to rely on the leading API discovery tools to locate the software that they need. In this section, we will list out some of these tools.

3.4.1. ProgrammableWeb

The ProgrammableWeb API directory has been in existence since 2005 - making it easily the oldest (and probably the most exhaustive) API discovery tool. Apart from open APIs and mashups, various types of relevant API-related stats can be obtained from here. ProgrammableWeb was acquired by Mulesoft in 2013.

3.4.2. Apigee

Apigee offers 'API controls' which enhance the discoverability of APIs. Interfaces developed by leading API providers (including Twitter and

Facebook) are available in the consoles. App-makers/API customers can easily learn API functionalities, before starting with the implementation. Methods and documentation types can also be chosen in the API consoles.

3.4.3. Mashape

This API discovery tool offers two-fold functionalities: as an advanced API management platform and as a generic API marketplace. Each API listed on Mashape is accompanied by a test functionality and detailed documentation, making things easier for clients. Both free and paid APIs are available on Mashape, and the community has well over 5000 Public APIs.

3.4.4. Mashery

The Mashery API Network is another excellent source for API discovery (Mashery was founded way back in 2006). In general, Mashery offers high-quality API management, technology and infrastructure support - with more than 150000 developers in the network. APIs are audited regularly (explaining the presence of the DX Certified badge on some APIs). The discovery tool includes Hacker League, Open Source Tools, and the in-house API Explorer section.

3.4.5. APIs.io

Unlike most other API discovery tools, APIs.io operates more like a smart search engine software for digital platforms. It has a fairly large number of web APIs - with the APIs.json format - in the repository. One of the biggest advantages of this tool is that, it does away with the need of manually adding APIs by contributors.

3.4.6. Google APIs Explorer

This one, as the name itself suggests, is a directory exclusively for Google APIs. Through this interactive and user-friendly too, users can browse through methods, place API requests and check out the responses. Documentations as well as supported methods and parameters corresponding to an API are displayed - when that API is selected. The Analytics API and the Gmail API are two examples of the APIs present in this tool.

3.5. Tools For API Testing

First, let us briefly explain the various types of API testing. As is the case for any other form of software, quality and performance are the principal points of concern among API clients. To ensure that an API delivers the service expected from it without any hitch, developers have to perform different API tests. We will briefly deal with them here.

3.5.1. Types Of API Testing

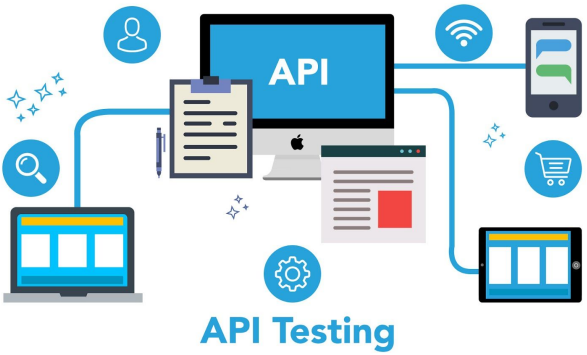
- **Usability Testing** - done to check the overall user-friendliness (read: developer-friendliness) of APIs.
- **Load Testing** - done to check the volume of callbacks/requests that APIs can handle at a time.
- **Functionality Testing** - done to check whether APIs are indeed functioning as expected/desired.
- **Proficiency Testing** - done to find out if APIs are helping app developers, and to what extent.
- **Security Testing** - done to ensure that the user authentication, access control,

permissions and other security parameters are working properly.

- **Reliability Testing** - done to check the consistency of API responses/outputs across different software/client projects.
- **Discovery Testing** - done to check whether APIs have adequate formal documentation for the users.

Let's move over to the API testing tools now. Most of the tools we will highlight are used to perform load/functionality/usability testing.

3.5.2. API Tools: Testing



3.5.2.1. HttpMaster

Primarily used to test web API calls (*i.e., load testing*), the HttpMaster tool automates the overall

process of web application testing. *POST, GET* and *DELETE* are some of the common *http methods* supported in this tool, along with a fairly large array of validation methods and expressions. API requests can be clubbed into batches with the dynamic parameters of the web development tool, making the testing process easier for developers. In addition to API testing, HttpMaster can be used for website testing and service testing as well.

3.5.2.2. SoapUI

One of the most '*complete*' API testing tools out there at present. From *load and regression testing*, to *compliance* and, obviously, *functional testing* can be done with this software resource. The built-in *Groovy* support enables API testers to generate complicated validation scripts with ease, while web method requests can be used to directly generate test cases. SoapUI offers cross-platform functionality and serves as a tool for *testing both REST APIs and SOAP APIs*. Assertions (*created with XQuery or XPath*) are used to generate the web method results. The test setup in SoapUI can also be altered, as and when required.

3.5.2.3. Postman

For manual API testing, Postman – which is basically a Google *Chrome plug-in* – can be just the perfect tool. Since Postman is, in essence, a high-end *HTTP client*, it supports practically all forms of modern web API data (*for extraction or exploration*). The interface of the tool allows testers to write out custom *Boolean test scripts*, while batches of REST calls can be created and saved (*for later execution*) too. A big advantage of Postman is that it is not a command-line based tool (*unlike, say, CURL*), which makes using it considerably easier.

3.5.2.4. DHC

Created by Restlet, DHC is a widely used *Web API testing* resource. The tool allows users to seamless integrate the API testing procedure with their *Continuous Integration (CI) and/or Continuous Delivery (CD)* delivery methods. The built-in graphical user-interface of DHC doubles up as an excellent *visual tool* for monitoring API calls. A large number of API requests can be bunched together in test scenarios, with the tool having the capability to handle requests of varying complexity. The responses to requests can be analyzed easily too.

3.5.2.5. Apache JMeter

A *Java-based*, multi-utility, *open-source* tool for load testing and functionality testing of the endpoints of Web services APIs. JMeter is increasingly being used to test *RESTful APIs* as well. The *multi-threaded* feature of this tool makes it ideal for performing effective, accurate load testing. Multiple *protocol types* are supported by Apache JMeter (*FTP, LDAP, SOAP/REST, HTTP/HTTPS, and more*), during performance testing and load testing of APIs.

3.5.2.6. vRest

For automated testing of *RESTful APIs* as well as *HTTP APIs*, vRest is a more than handy online tool. Depending on the precise specifications of each API, documentations are generated by the tool – and it also delivers high-speed validation services for REST APIs. The *Mock Server Functionality* of vRest, which allows smooth creation of API mocks, also deserves a special mention. Data can be imported from the *Swagger API framework* without any hassles. vRest also comes with *JIRA-Jenkins integration.*

3.5.2.7. Parasoft

Parasoft is often the go-to testing tool for APIs without graphical user interfaces (*GUIs*). A vast range of protocols is supported by the Parasoft interface, making the task of specifying automated test scenarios a lot simpler. Scenarios of varying complexities are automated by this tool – *across mainframes, databases and even messaging layers*. The tests created with Parasoft are typically *reusable and scalable, apart from being easy to maintain*. The tool is best for performing *regression testing of APIs* with state-of-the-art validation methods. A high point of Parasoft is that, it lets users create tests without having to actually code.

3.5.2.8. TestingWhiz

Yet another code-free API testing tool (*it also performs mobile testing, big data testing and database testing*). Regression tests for APIs can be easily automated with TestingWhiz, and the tool also offers reliable *automated web UI testing* services. Practically all the popular browsers are supported (*the likes of Firefox, Chrome, IE, Opera, Safari*) – enhancing the coverage of tests as well as the convenience of users. More than *290 commands* are available for generating modular automation scripts – doing away with the need for coding.

HP QTP and TestMaker are a couple of others powerful tools for API testing. Thorough, careful testing is an absolute success for paving the way to success for an API...or for any software, for that matter.

3.6. Apigee: The Smart API Management Platform

Launched in 2004, Apigee has gone on to become a widely used company for managing application program analyses. It offers two products: the first is Apigee Edge, which serves as a cloud-hosted or locally-hosted API management platform. The other is Apigee BaaS - a platform for backend-as-a-service solutions.

The Apigee platform is used by web/mobile app developers around the world for designing, scaling, analyzing and even securing digital interfaces. Some key functionalities of Apigee have been highlighted here:

- Creation of scalable, state-of-the-art RESTful APIs from SOAP services.
- Deployment of API proxies from Swagger specifications or Open API specifications.
- Centralized and secure management of APIs.

- Caching, rate limiting and dynamic routing for API traffic management.
- Application of SQL/XML threat protection and similar policies.
- Inclusion and maintenance of SAML, OAuth 2, PCI/HIPAA and 2-way TLS data compliance standards.
- Providing tier-based structure of APIs to user.
- Driving up user-engagement levels.
- Making in-depth insights, stats and figures on API developer analytics available.
- Monitoring errors, response times, traffic volumes and other key metrics.
- Visual troubleshooting by recording processing steps.
- Integration of API lifecycle management in Systems Development Lifecycle (SDLC).
- Auto Scaling to handle callback spikes.
- Customized model building and monetization solutions.
- API call analysis to block out ‘bad bots’.
- Support for REST APIs for Internet of Things (IoT) services.

The Apigee Edge platform is used to create and manage API proxies. The platform is mostly used by software developers who need a secure, easy-to-use and multi-featured backend resource.

3.7. More API Tools

In addition to the ones highlighted in the previous sections of this ebook, there are several other useful API tools available to developers. Let us briefly learn about a few of them over here.

3.7.1. KONG

A powerful open-source API management tool with extensive documentation and an active community on Github. Rate limits, OAuth support and SSL termination are some of the main features of KONG - which has been built on openresty. REST APIs are used to create modules in KONG.

3.7.2. Galileo

Formerly known as APIAnalytics, this tool displays the usage stats of APIs to the coders. Right from the nature and types of API errors, to the endpoints that are being utilized (along with the associated user-distribution) - all API-related metric can be monitored with this tool. Galileo does away with the need for manually parsing logs - and that’s a big advantage for API providers.

3.7.3. APIBoard

Another well-rounded documentation tool that helps programmers track all the recent changes/modifications made in APIs. APIBoard also creates series of API calls - also known as ‘workflows’ that operate sequentially to complete tasks.

3.7.4. APIGarage

A multi-platform API console made on top of Angular.js and Electron that ensures seamless workflow integration for developers - providing a boost to the productivity levels of the latter.

3.7.5. Blockspring

Blockspring focus on ramping up consumption and usage figures of APIs, by making them accessible through common desktop applications (for instance, MS-Excel). The tool creates customized API blocks which can be run through different business apps - with customers not required to have prior programming knowledge.

3.7.6. APIStudio

Swagger specifications can be used to generate APIs, in this graphical-user-interface for API development. Client libraries and server-side codes can also be generated with APIStudio.

3.8. API Authorization Tools: OAuth 2

Data security is of paramount importance on digital platforms. APIs typically contain big data from the cloud or from other third-party sources - and there simply has to be a reliable, user-friendly, authorization framework that offers selective access through HTTP services, to permitted user-accounts. The tool AUTHORIZES third-party applications to access user accounts AFTER delegation of user-authentication to the host service. Github and Facebook are two examples of the HTTP services supported by the OAuth framework.

3.8.1. Moving on from OAuth 1

OAuth 2 has several interesting points of difference from its predecessor - the 'AuthSub' and 'FlickrAuth' - based OAuth 1 framework. For

starters, the latter required ‘client ID’ and ‘client secret’ for signing requests - while OAuth 2 removes the confusion by making the use of HTTPS mandatory for all interactions between APIs, clients and browsers. Scalability is also something OAuth 2 performs much more efficiently than OAuth1 (the temporary signing credentials and the state management used to make things difficult). The API server no longer has to access the ID and the secret of the concerned application(s). Also, OAuth 2 delivers a significantly better experience on native mobile applications (OAuth 2, in contrast, was optimized for web browser apps).

3.8.2. OAuth User Roles

The OAuth 2 framework has four separate, well-defined roles:

Authorization Server - The ‘user’ either accepts or declines API requests on the authorization server. In certain setups, the API server itself serves as the authorization server (typically for small-scale applications).
Client - Within the OAuth framework, the external application that requests for user information/user account access is referred to as the ‘client’. Requires prior user permissions.
Resource Server - In setups where the API server and the Authorization Server are not one and the

same, the former is known as the resource server. It is used to obtain the requested user-information.
User/Resource Owner - As the role name itself suggests, this is the person or entity from whom the API requests information.

3.8.3. OAuth 2: Grant Types

A third-party app can use different methods to request information. The manner in which this is done determines the type of grant provided by OAuth 2 to the final user. The framework offers as many as 4 different types to users. Let's quickly check them out here.

Implicit grant - Typically given to apps running on third-party applications (web/mobile).
Authorization Code grant - This grant type typically goes with server-side apps.
Client Credentials grant - Provided with the API access of the applications, and
Password - This OAuth grant type allows resource owners to log in with their unique username and password combinations.

UMA, Open ID Connect and Green Button are the three main protocols supported by OAuth 2.

3.8.4. OAuth 2: Registering An Application

Users have to fill up the registration form (*available from the website*). Apart from the name of the app and its website (*along with, maybe, a short description*), they also have to specify a ***Redirect URL*** (where the app will redirect to, after the access permission is granted). Post-registration, a ***Client ID*** and a unique ***Client Secret*** are allotted to every user.

3.8.5. OAuth 2: Access Tokens

Tokens, or 'access tokens', are used by the client-side applications to access and utilize information from user-accounts (*a form of permissions*). The tokens are generated by the server, following a validated request by the concerned third-party app.

From Facebook and Twitter, to Yahoo, GitHub and Google – most of the big players rely on Oauth2 to provide information from user-accounts on their servers. The framework is easy to use, the registration procedure is straightforward, and it has top-of-the-art security features.

3.9. API Practices

Software development is tricky business. Developers can place one foot wrong while coding, and the resultant bugs and issues might become too large to resolve. Not adhering to the specified quality parameters is likely to prevent APIs from functioning as desired (i.e., the functionality testing will fail). In this section, we will look around at a few important API practices.

3.9.1. API Best Practices: Designing

(Note: We will focus primarily on RESTful APIs here, as we have done throughout this book)

3.9.1.1. Using Nouns Instead Of Verbs in URL

Unlike SOAP APIs, *RESTful APIs make data available as 'resources'.* In such a resource-oriented ecosystem, all elements of the API domain (*right from Sales and Clients, to Orders, Documents and Users*) should be considered as separate '*entities*'. This, in turn, means that nouns – and not verbs – have to be used in the URL (*e.g., POST abc.com/trainings instead of POST abc.com/gettrainings*). Verbs can be delegated systematically with the HTTP verbs. Each resource needs to be exposed by an API, doing away with the difficulties of actually making endpoints of each specific '*action*'. The HTTP verbs will indicate the action to be performed, while the resources as nouns in the URL will highlight the purpose/functionality.

3.9.1.2. Communication Format in HTTP Folder

The format of communication should be clearly mentioned in the HTTP header of a REST API. The *serialization formats* help both the API clients as well as the API developers to understand how communications/network requests would take place through the interface. For listing out all the supported response formats, use *'Accept'.*

'Content-type', on the other hand, specifies the exact format of the request.

3.9.1.3. Application of HATEOAS Principle

A factor that can trip up an otherwise good REST API is an over-complicated navigation scheme. HATEOAS, or *Hypermedia As The Engine Of Application State*, is a principle that addresses this issue well – and hence, should be utilized during the API designing phase. In essence, the HATEOAS approach is all about placing properly working (pre-tested) *hypertext links to facilitate smooth navigation* within the interface.

3.9.1.4. Two URLs For Each Resource

API designers should create 2 separate base URLs corresponding to each resource. *One of them will be handling specific (single) values, while the other will be for managing multiple values.* This approach helps users keep track of their network requests on a real-time basis. Relying on a single URL for both specific and multiple values increases the chance of errors being returned.

3.9.1.5. Using subresources in REST APIs

You need to keep your APIs simple, and introducing *subresources* is a great way of doing that. This is particularly important for interfaces that involve a large number of relationships – where using only the top-level APIs can be complicated. In general, *any resource that is a part of another (parent) resource can be used as a subresource.* These subresources offer two-fold benefits: firstly, within the representation of the resource, the *dependency on individual keys is brought down*. Also, API customers can *understand the resources more easily*, when subresources are present. To put it simply, subresources significantly enhance the readability of RESTful APIs.

3.9.1.6. Plurals in Endpoint Names

While using plurals for every single resource does not seem correct from a strict grammatical point of view – there are other key advantages of using them over the singular forms (*and API developers need not be overtly bothered about grammar in their codes any way*). The use of plurals (e.g., api/users instead of api/user) gives a clear indication of the *entire collection of data* that needs to be fetched (*in our case, 'user'*). Barring a few exceptions, plurals should be used for most resources.

3.9.2. API Best Practices: Security

3.9.2.1. SSL Certificate Validation

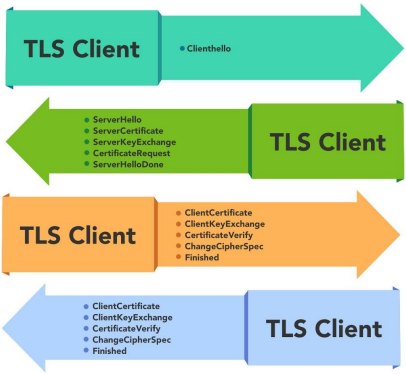
For ensuring the security of web APIs and clients, validation of concerned *SSL certificates* is essential. However, developers often make the folly of making this validation process rather half-baked – as a result of which hackers can issue their very own bogus certificates (*all that's required is a working net connection*), get random users to validate on these, and access confidential user information (*breaking into the the encrypted data transfer process*). *Malicious API traffic interception and the 'weak validation' on fake certificates can deliver API keys, passwords and usernames right in the hands of hackers* as well. Set up the data encryption and validation methods based on the

sites that the web client will access. For iOS applications, consider *'key pinning'* to keep things more secure.

3.9.2.2. JSON/REST instead of SOAP/XML

Most vulnerabilities from the server-side can be easily tracked and fixed in SOAP (*Simple Object Access Protocol*) APIs. However, the XML data format – which is bundled in with the SOAP protocol – has several *'soft targets' for hackers*, like *Denial of Service (DoS), attacks by external entities*, and even problems in *XML encryption*. The fact that the SOAP protocol remains in production for extended periods due to heavy system dependencies complicates matters further. Unless you are working on a corporate API or a software legacy system, it makes a lot of sense to ditch SOAP/XML *in favour of the much simpler JSON/REST* (Representational State Transfer) combination. The potential security threats will be much lower.

3.9.2.3. Transport Layer Security



Absence of a *Transport Layer Security (TLS)* in an API is practically equivalent to handing out open invitations to hackers. Transport layer encryption is one of the most elementary '*must-have-s*' in a secure API. Unless a TLS is used, risks of the fairly common *'Man-In-The-Middle' attacks*(where an unauthorized third-party can intercept the transferred data and modify it) remain very high. Use both SSL and TLS in your APIs…they are neither too pricey or too complicated, and they go a long way in removing basic API vulnerabilities.

3.9.2.4. Degree of control to customers

As soon as network requests (*API calls*) start coming in, the API gets exposed – and if you let customers use your APIs in whatever way they like – a hack attack might be waiting just around the corner. Many APIs do *not set any specific password complexity rules, do not track the API metrics, or allow repeat session ID tokens*. Remember that while most of the users are genuine, there can always be a handful of miscreants, looking to use your API to illegally usurp user-data and maybe even introduce bugs in the system (*the 'black-hat hackers'*). Set *limits on concurrent API connections*, implement password length/complexity requirements, make *re-authentication mandatory* for extended usage, and be very careful while analyzing the API usage metrics. If any suspicious activity is detected, find out the reasons behind it.

3.9.2.5. Maintenance of high coding standards

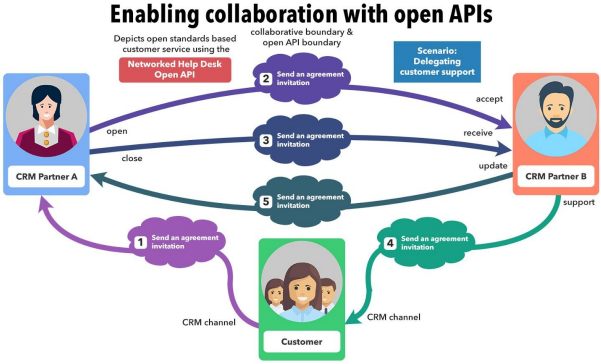
API developers need to refrain from taking the short way out by following the so-called *'Just Enough Coding'* route, which might expose API(s) to serious vulnerabilities. The risk might not be apparent at first, since the concerned API might be doing okay on the usability and the functionality

fronts. However, poor coding standards *will not allow proper integration of API(s) with the overall platform/ecosystem*. Problems can arise from multiple errors – right from *absence of 'type checking'* (which allows uploading of any file and deployment of the app on the server) and *memory overflows*, to incorrect *error handling methods* and even choice of the 'wrong' language for coding. Coders need to take time out to get a hang of how APIs are meant to work and how customers would use them. That will help in coding properly.

3.9.2.6. Robust security at endpoints

API developers should ideally include *'key signing', 'hashes', 'shared secrets'* and other such common endpoint hardening technologies within the API development cycle. Making changes at a later stage is difficult – since the legacy systems which use the API might suffer performance issues. In case the *endpoint hardening* is not done during the early phases of API development, it might not get done at all. As a result, the endpoints will remain vulnerable – and any competent hacker will have a field day.

3.9.3. API Best Practices: Enterprise APIs



Around 85% of the large enterprises across the globe will have proprietary API programs by the end of 2018. The constant advancements in inter-organizational collaboration standards and information transfer technologies are fueling the growth of enterprise APIs in a big way. In this section, we will discuss the best practices of handling such API systems, to get the best out of them.

3.9.3.1. Choice of the 'correct API tier'

Enterprise API management is much more than creating a standard, '*one-size-fits-all*' API, and then hoping that it would work like a charm for all

companies. The onus is on business entrepreneurs to *select the custom API tier and strategy*, that would enable their firms to *create more value internally, and deliver more value to customers* and other stakeholders. *Startups should ideally go for the Open API tier*, which can provide them with large-scale business recognition and exposure. *More established enterprises*, on the other hand, can have a *combination of a customer-only API tier* (that has all important transaction data) *and an internal API tier* (with the proprietary, confidential internal information). Since Open APIs can be accessed and used by anyone, they can somewhat dilute the brand image of larger companies.

3.9.3.2. 'API-first' design

In a service-oriented architecture (*SOA*) system, this is no longer a debate. It is almost *mandatory for API providers to adopt an API-first design approach*. In essence, this means creating a smooth, functional and powerful interface to start with, and then hooking it to the backend logic setup (*the traditional approach puts more importance on the backend, and relegates APIs to almost an afterthought*). The API-first approach works well on two fronts – firstly, *API testing* becomes a lot less complicated, and secondly, the *API implementation* process remains thoroughly defined and documented. A backend-first model muddles the very existence of enterprise APIs.

3.9.3.3. Connecting with external ecosystems

Buyer behaviour is hard to predict. There also remains considerable uncertainty regarding demand fluctuations over time too. In order to deal with these, more and more companies are using *APIs that open up their information systems to external ecosystems*, comprising of third-party coders, mobile app developers, testers, analysts, and the like. As a result, all the necessary experiments and surveys can be conducted with the help of the information assets at the disposal of companies. The degree of third-party information access can be controlled too. Firms that open up their ecosystems with APIs typically share their revenue-streams with third-party developers.

3.9.3.4. Need for enterprise APIs

In a *Layer 7 survey*, it was revealed that *72%* of the respondents created APIs to *bolster the performance of their in-house mobility systems*. This was closely followed by the demand for *better integrations with business partners via APIs* (70%). Over 65% enterprises expressed their willingness to *develop cloud-based APIs*. Interestingly, around 55% of the respondents wished to *create third-party app developer communities with APIs*.

3.9.3.5. Availability

Enterprise APIs need to be simultaneously *available on-premise* as well as *on the cloud* network. This enhances overall usability, since users are able to access additional resources (*as and when required*) without any problems. What's more, switching over from the on-premise API model to the cloud API model also becomes easier. The key lies in ensuring that the enterprise API system *can operate on both the platforms without requiring any modifications*.

3.9.3.6. Monitoring API Effectiveness

Like any form of software, APIs can also fail. If persisted with, a buggy API can cause significant loss of revenue for businesses. That, in turn, brings to light the value of *analysing the available API metrics* on a regular basis, to track progress and assess results/outputs. By tracking the analytics/metrics of an enterprise API, two things can be monitored: a) the API itself is working as it is meant to (*the technical perspective*), and b) the API is indeed creating innovations and generating value internally and externally (*the business perspective*).

3.10. API Best Practices: Strategy Optimization

On average, 7 out of every 10 enterprises have already implemented customized API strategies. APIs have also emerged as important income-generating platforms, with nearly 50% of these enterprises earning money from digital platforms. Here are some key points in the API strategy optimization exercise.

3.10.1. Understanding the API Value Chain

For make your API strategy successful, you need to know the flow in which APIs are used. This is referred to as the *'API value chain'*. It has the existing *backend systems*/enterprise IT systems at the first block. Next up in the chain are the *API providers*, who create and deliver the interfaces to the web/mobile *app developers*. The latter use the APIs to create *client applications*. These, in turn, are downloaded and used by the *end-users* or the general public. So, the 'API value chain' looks something like this:

Backend architecture → API Providers → Application Developers → Client Apps → Final Users

3.10.2. Importance of API Version Control

Having an API strategy without a *version roadmap* is like trying to control a rudderless ship. There can be heightened security threats, app makers can start using them to make applications (*and make changes in the API themselves*), and outdated APIs can get dragged along for too long. As the API entrepreneur, the onus is on you to settle on a *version control system for your application program interfaces*. Plan how your APIs will be tracked and monitored (consider all the measurable metrics), what the security protocol and related updates would be, and how you will retire/phase out the older APIs. For that, you will also need to have an idea about the four stages in an *API lifecycle:*

API Analysis → API Development → API in Operations → API Retirement

3.10.3. A 'Minimum-Viable-Product' (MVP)

A MVP is something like a *beta release* of an API. It is never advisable to release the final version of an API and then make changes in it (*that affects the way in which the APIs are used by those who make apps as well, diluting the overall scenario*). Instead, a *MVP (Minimum Viable Product) version of your APIs* should be released as quickly as possible. Make sure that it is *accompanied with proper formal documentation, terms of use, a user-friendly*

sandbox/public endpoint, and a robust security setup. The MVP can be initially offered to in-house developers, before it is rolled out to business collaborators, who can become customers of the APIs. The feedback received has to be monitored, the required changes implemented, and then the full-blown version 1.0 of an API is released.

3.10.4. Design Style Selection

Designing an API that would deliver value to your enterprise is not the easiest task in the world, and going with a wrong design model can further complicate matters. Based on your business goals and pre-specified API visions (*i.e., how APIs are going to boost your business operations*), you can choose a *Pragmatic REST model*, a *True Rest (Hypermedia) design model*, or a *Web Service Tunneling model*. With the importance of Internet of Things (IoT) rising, many enterprises also opt for the *Event-driven API designing* style. Make sure that the API design strategy you pick will make the interfaces in sync with your backend systems.

3.10.5. API Gateway

An API strategy might be good in theory, and it might even have a captive developer audience (*a ready set of customers*). However, all the good work might be undone, if API developers are not careful while creating the *'API Gateway'*. The

gateway is supposed to *deliver all the functionalities included in the core API infrastructure* – right from orchestration and caching, to data security and access control. For an API program to be usable, its gateway needs to be very carefully developed.

3.10.6. User roles in API Program

It's all very fine to have an API strategy on paper. However, it's an entirely different ball game to actually implement it within your enterprise. Not engaging the right personnel in the right roles in API management is one of the biggest reasons for the failure of many enterprise APIs. The *enterprise architect(s) should be working as the lead managers* in an API strategy, overseeing the development, designing, modifications and deployment of APIs (*along with their backend integration*). *Product managers should double up as the connection/communication channel* between the ecosystem of API developers and API customers. The responsibilities of *maintaining the API architecture should be taken by the system administrator(s)*. Other important team players involved in the API program include *API testers, senior software engineers, internal/external app developers, and other stakeholders*.

4. APIs In Various Industries

In this chapter, you will learn:

- The implementation of APIs in different fields (including examples)

The rapid growth of the 'API economy' has left its mark on different industrial sectors and business activities around the world. APIs are no longer something that are dealt with only by the most tech-savvy companies. Instead, they are increasingly becoming mainstream, with enterprises from various sectors moving on to digitized platforms for implementing innovation and cutting down on overall operational expenses. In this section of the ebook, we will briefly take a look at the industrial sectors in which APIs find widespread acceptance.

4.1. Banking Industry

APIs are extensively used in mobile banking apps, offering real-time account information and the convenience of doing secure transactions. Digital support have also boosted e-payments (via credit card or debit card), as well as the online promotional campaigns undertaken by banks. APIs have a strong role to play in Partner Loyalty Programs too.

Examples: Tripit Point Checker, PayPal

4.2. Media & Entertainment Industry

Key stakeholders (e.g., film producers) utilize APIs to facilitate ticket-buying for the general users. In addition, access-information on the basis of tiers is also provided digitally. Audio recordings are tracked with APIs as well, and the latter is also used for product placement advertisements within programs.

Examples: Comingsoon.net, Moviefone

4.3. Travel Industry

The scopes for developing custom APIs for the travel and navigation industry are immense. Right from direction and route assistant apps, to software for inventory management and order processing in airlines - APIs are extensively being used by web/mobile developers worldwide. Travel aggregator apps also typically rely on backend cloud support for full functionality. Specialized smart car APIs are also growing in popularity.

Examples: Tripit, Google Maps

4.4. Retail Industry

Locating stores nearby (in any particular location), keeping track of the status of inventories, and sharing the details on the latest offers, discounts & special deals are some of the important functions facilitated by APIs and mobile apps. Product catalogs can be created/digitally updated real-time with the help of APIs too.

Examples: Amazon, C-Net

4.5. Medical/Healthcare Industry

APIs are increasingly being used in pharmacies for order entry and processing. A digital platform also makes two-way communication (via calling or messaging; between practitioner and patients) more seamless and more secure. Medical stores/clinics can access their appointment books with the help of apps powered by BaaS.

Examples: WellDoc Diabetes Manager, Merck Medicus

4.6. Energy & Utilities Industry

With connected homes and internet of things (IoT) progressing rapidly, APIs are becoming integral elements in the development of apps for this sector. Digital platforms are revolutionizing the way in which daily energy consumption figures are tracked and managed. Field repair departments find it handy to use optimized software to maintain inventories. APIs are also being used to track and manage service requests (new and existing).

Examples: Southern California Edison, MobileIron SmartHome

4.7. Telecom Industry

The pattern and purposes of API usage in the telecom industry has a lot in common to how such digital interfaces are used in the banking industry. Once again, APIs facilitate secure fund withdrawals/transfers (through credit/debit cards), and make it easy for people to access account information on the go. Many telecom players launch advertising campaigns through mobile applications too.

Examples: PayPal, GroupOn

4.8. Automobile Industry

The connected car segment is one of the fastest growing sectors in the grand IoT scheme worldwide. APIs play multifarious important roles over here. Apart from traffic alerts, navigation assistance and other location-based services, APIs are being used to access different instances of service records from the cloud (as and when required). Mobile apps backed by custom APIs also send/receive alerts to/from vehicle dealers.

Examples: BMW Connected Drive, Google Maps

4.9. Insurance Industry

With the help of API-based digital solutions, insurance companies are being able to deliver higher customer-satisfaction levels than ever before. People can now easily upload claim-related images, know more about company policies, and get premium deposits in bank accounts approved - all via the mobile/digital medium. Information exchange regarding repair claims and coverage is also facilitated.

5. API Stories

In this chapter, you will learn:

- Six API use cases from the real world (Case Studies)

5.1. Expedia - Smarter Marketing

Expedia is a leading US-based online travel and reservation company, with a vast partner/affiliate network. It constantly strives to deliver greater value to all customers, through continuous innovation, technological advancements and provision of high-quality business/leisure travel options. Prior to 2010 (when Expedia implemented Amazon Web Services), the company used to do business with an website including HTML frames. The partners had to embed elements in the site. The process was long-drawn, and at times, cumbersome.

Enter APIs, and the day-to-day operational processes at Expedia changed for the better in a

big way. Currently, more than 90% of the company's annual revenue is derived from these digital platforms alone. The cloud virtualization strategy helped Expedia tie up with more partners (for instance, travel aggregator portals and hotels). These partner websites had linkbacks to Expedia - driving up the business for the latter. The marginal cost figures for Expedia (for the incremental business) also remains low.

The key advantages Expedia has obtained by using APIs can be summarized as under:

- Greater reach to global clients, with accurate and updated information.
- Superior platform scalability.
- Smarter information processing.
- Making app development cycles shorter.
- Quicker bug-fixing and troubleshooting.

5.2. Facebook - Better Partnerships, Lower Costs

Facebook Connect, the social media website's authentication APIs, has helped FB establish and maintain a robust, cost-effective and steadily increasing presence. From the users point of view, the Facebook Connect platform offers the now-familiar 'Log in with Facebook' option (on other applications/websites). This feature goes a long way in establishing a universal set of credentials for most web resources.

Strong integrations with mobile platforms and third-party apps is a high point of the Facebook Connect API platform. Users can use their Instagram accounts to publish images on FB, as well as chat with their contacts. The platform is also embedded with the iOS platform (since iOS 6), and offers several handy features - collaboration with native Apple apps, contact information synchronization etc - as well.

The key advantages Facebook has obtained by using APIs can be summarized as under:

- Establishing universal IDs for users.
- Integration with third-party web/mobile applications and websites.

- Significant cost reduction. In the absence of APIs, Facebook would have required well over 1500 marketing personnel, and a budget in excess of $60 million (in three years).
- Easier onboarding for third-party app developers.
- Seamless partnerships.

5.3. Nike - The Nike+ Story

One of the biggest sports apparel and footwear producers in the world - Nike - owes a lot of its recent success to its multi-featured digital platform, Nike+. The latter is a activity and fitness tracker, originally unveiled in 2006 - and currently available for both iOS and Android platforms (including wearables). Putting things in perspective, the Nike+ platform has been instrumental in significantly enhancing the brand's visibility and recall value in the customer-mindspace.

Another ongoing benefit that this API-powered platform offers is a smarter strategy to establish business partnerships. Integration with corporate applications has also been rendered a lot simpler. The data collected in the Nike+ platform can be uploaded directly to the company site, through iTunes or other external programs. The

Nike+Running application is also available in stores.

The key advantages Nike has obtained by using APIs can be summarized as under:

- ❖ Ensuring greater reach and exposure of the Nike brand.
- ❖ Smooth tracking and transfer of activity data.
- ❖ Better collaboration, partnership and networking opportunities.
- ❖ Easy implementation with external apps (leisure/corporate).

5.4. Salesforce - CRM Services, Revolutionized

American cloud computing firm Salesforce is one of the pioneers, when it comes to establishing an API strategy for business. Through custom-built APIs, the company manages to provide a large bouquet of important CRM services to users from all over. These services include social media tracking, automation of sales force tasks, performance management and helpdesk support activities, among others.

Via migration to digital platforms, Salesforce has found it progressively easier to collaborate with

other firms/partners. Implementation of the Salesforce software can be directly attributed to the smooth implementation in Google, Microsoft, Oracle and SAP applications.

The key advantages Salesforce has obtained by using APIs can be summarized as under:

- Like Facebook Connect, the Salesforce APIs have helped in serious cost-reduction.
- Smooth implementation with third-party solutions.
- Wide array of automated, high-quality CRM services.
- Constant innovation and service quality improvement.

5.5. Netflix - Efficient Database Management

Netflix, the popular movies and television show streaming service, presents yet another successful use case of API strategy implementation. The service migrated to API platform back in 2008, and since then, have reaped considerable benefits from the data virtualization and cloud support. By 2012, the Netflix API was being used by well over 20000

third-party app developers - while media content was supported by more than 800 devices.

The arrival of Netflix API helps developers in three important ways: first, for accessing the rental history of the video database; next, for managing the constantly expanding database resource; and finally, for handling queue-management related tasks. With the help of the API platform, the developers can easily build custom applications for devices - without Netflix having to incur additional expenses. If there was no API, there would have been no third-party app makers - and the company would have had to cough up more than $1 billion annually, to develop everything in-house.

The key advantages Netflix has obtained by using APIs can be summarized as under:

- Avoiding high development costs by allowing access to third-party developers.
- Smarter management of the Netflix database.
- Increased reach, through compatibility with more devices (with apps).
- Commercial use of the free Netflix API.

5.6. Coca Cola - More Fizz To Business

In 2011, Coca Cola launched its much-talked-about Information Transparency Policy, and that marked the start of the soft drink conglomerate using the API platform for business. The company placed prime importance on establishing the essential security protocols first. That, in turn, ensured, that APIs could be churned out without any hitch later.

Following a streamlined API strategy has been instrumental in helping Coca Cola get more out of its core strategic capabilities, through constant innovation, quality improvement, higher productivity levels, and improved time-to-market. Over the years, multiple departments have started to rely on APIs for operations - right from procurement and supply-chain management, to sales, finance and IT.

The key advantages Coca Cola has obtained by using APIs can be summarized as under:

- Speeding up the overall development by bringing down workloads.
- Helping cloud and mobile project management teams.
- Significant internal benefits (i.e., for in-house developers).

- ❖ Opening up the business and making information readily accessible.
- ❖ Optimal utilization of digital capabilities.
