**API – Application Programming Interface**

This is a connection between computers or between computer programs.

It is a type of software interface, that offers a service to other pieces of software.

A document/standard that describes how to build such a connection or interface is called an *API specification.*

A computer system that meets this standard is said to *implement* or *expose* an API.

The term “API” may refer either to the specification or to the implementation.

An API is not intended to be used directly by a person (end user) other than a computer programmer who is incorporating it into software.

An API is often made up of different parts which act as tools/services that are available to the programmer and a programmer/program that uses one or more of these parts is said to *call* that portion of the API.

The calls that make up the API are also known as *subroutines, methods, requests* or *endpoints.*

An API specification *defines* these calls: it explains how to use/implement them.

An API is commonly used to hide the internal details of how a system works, exposing only those parts a programmer will find useful and keeping them consistent even if the internal details change later.

An API may be custom-built for a particular pair of systems, or it may be a shared standard allowing for interoperability among many systems.

The term “API” is often used to refer to *web APIs,* which allow for communication between computers that are joined by the internet.

There are also APIs for programming languages, software libraries, computer operating systems and computer hardware.

**Purpose of an API:**

Generally, an API opens a software system to interactions from the outside.

This allows two software systems to communicate across a boundary (*an interface*) using mutually agreed-upon signals: connects two software entities together.

Unlike a user-interface (UI), an API is not usually visible to users since it is used for machine-to-machine communication.

A well-designed API exposes only objects or actions that are needed by a software or by software developers: it hides details that have no use (*abstraction*).

Building software using APIs has been likened to using “building-block toys” since software services are analogous to the bricks: they may be joined together via their APIs, composing a new software product through a process called *integration.*

An example is that of a weather sensor that offers an API. When a certain message is transmitted to the sensor, it will detect the current weather conditions and reply with a relevant weather report. The message that activates the sensor is an *API call*, and the weather report is an *API response.*

An API is often compared to a *contract*. It represents an agreement between parties: a service provider who offers the API and the software developers who rely upon it.

**Types of APIs:**

1. **Libraries and Frameworks**

The interface to a software library is one type of API. The API describes and prescribes the “expected behaviour” (*a specification*) while the library is an “actual implementation” of this set of rules.

A single API can have multiple implementations (*or none, being abstract*) in the form of different libraries that share the same programming interface.

The separation of the API from its implementation can allow for programs written in one language to use a library written in another. For example, because Scala and java compile to compatible bytecode, Scala developers can make use of any Java API.

API use can vary depending on the type of programming language involved.

For procedural languages such as Lua, an API could consist of primarily basic routines to execute code, manipulate data or handle errors. For an object-oriented language, such as Java, the API would provide a specification of classes and its class methods.

Hyrum’s Law – *with a sufficient number of users of an API, it does not matter what you promise in the contract, all observable behaviours of your system will be depended on by somebody.*

Several studies show that most applications that use an API tend to use a small part of the API.

*Language bindings* are also APIs. By mapping the features and capabilities of one language to an interface implemented in another language, a language binding allows a library or service written in one language to be used when developing in another language.

An API can also be related to a software framework: a framework can be based on several libraries implementing several APIs, but unlike the normal use of an API, the access to the behavior built into the framework is mediated by extending its content with new classes plugged into the framework itself.

1. **Operating Systems**

An API can specify the interface between an application and the operating system. For example, Microsoft has a backward-compatible API, particularly within its Windows API (*Win32*) library, so older applications may run on newer versions of Windows using an executable-specific setting called “*Compatibility Mode*”.

An API differs from a ABI (*Application Binary Interface*) in that an API is *source-code based* while an ABI is *binary based.*

1. **Remote APIs**

Remote APIs allow developers to manipulate remote resources through protocols, specific standards for communication that allow different technologies to work together, regardless of language or platform.

For example, the Java Database Connectivity API allows developers to query many different types of databases with the same set of functions, while the Java Remote Method Invocation API uses the Java Remote Method Protocol to allow invocation of functions that operate remotely, but appear local to the developer.

Therefore, remote APIs are useful in maintaining the object abstraction in object-oriented programming; a method call, executed locally on a proxy object, invokes the corresponding method on the remote object, using the remoting protocol, and acquires the result to be used locally as a return value. A modification of the proxy object will also result in a corresponding modification of the remote object.

1. **Web APIs**

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets, which also is a Service Level Agreement (SLA) to specify the functional provider and expose the service path or URL for its API users.

An API approach is an architectural approach that revolves around providing a program interface to a set of services to different applications serving different types of consumers.

When used in the context of web development, an API is typically defined as a set of specifications, such as Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

An example might be a shipping company API that can be added to an eCommerce-focused website to facilitate ordering shipping services and automatically include current shipping rates, without the site developer having to enter the shipper's rate table into a web database

Web APIs allow the combination of multiple APIs into new applications known as *mashups.*

In the social media space, web APIs have allowed web communities to facilitate sharing content and data between communities and applications. In this way, content that is created in one place dynamically can be posted and updated to multiple locations on the web.

**API Design:**

The design of an API has significant impact on its usage. The principle of information hiding (*abstraction*) describes the role of APIs as enabling modular programming by hiding the implementation details of the modules so that users of modules need not understand the complexities inside the modules.

Thus, the design of an API attempts to provide only the tools a user would expect.

The design of APIs represents an important part of software architecture: the organization of a complex piece of software.

**API Release Policies:**

The main policies for releasing an API are:

1. **Private –** such an API would be for internal company use only.
2. **Partner –** only specific business partners can use the API. For example, taxi companies like Uber can allow approved third-party developers to directly order rides from within their apps. This allows the companies to exercise quality control by curating which apps have access to the API.
3. **Public** – the API is available for use by the general public. For example, Microsoft makes the Windows API public so that the software can be written for their platform. Not all public APIs are generally accessible by everybody. ISPs like CloudFlare use RESTful APIs to allow customers and resellers access to their infrastructure information through granting API tokens or customer status validations.

**Public API Implications:**

An important factor when an API becomes public is its "interface stability". Changes to the API—for example, adding new parameters to a function call—could potentially break compatibility with the clients that depend on that API.

When parts of a publicly presented API are subject to change and thus not stable, such parts of a particular API should be documented explicitly as "unstable". For example, in the Google Guava library, the parts that are considered unstable, and that might change soon, are marked with the Java annotation *@Beta.*

A public API can sometimes declare parts of itself as *deprecated* or *rescinded*. This usually means that part of the API should be considered a candidate for being removed, or modified in a backward-incompatible way. Therefore, these changes allow developers to transition away from parts of the API that will be removed or not supported in the future.

Client code may contain innovative or opportunistic usages that were not intended by the API designers. In other words, for a library with a significant user base, when an element becomes part of the public API, it may be used in diverse ways e.g. Black-hat hacking.

**API Documentation:**

This describes what services an API offers and how to use those services, aiming to cover everything a client would need to know for practical purposes.

Documentation is crucial for the development and maintenance of applications using the API. API documentation is traditionally found in documentation files but can also be found in social media such as blogs, forums, and Q&A websites.

Traditional documentation files are often presented via a documentation system, such as Javadoc or Pydoc, that has a consistent appearance and structure. However, the types of content included in the documentation differs from API to API.

In the interest of clarity, API documentation may include a description of classes and methods in the API as well as "typical usage scenarios, code snippets, design rationales, performance discussions, and contracts", but implementation details of the API services themselves are usually omitted. It can take a number of forms, including instructional documents, tutorials, and reference works. It'll also include a variety of information types, including guides and functionalities.

Restrictions and limitations on how the API can be used are also covered by the documentation. For instance, documentation for an API function could note that its parameters cannot be null, that the function itself is not thread safe.

Because API documentation tends to be comprehensive, it is a challenge for writers to keep the documentation updated and for users to read it carefully, potentially yielding bugs.

API documentation can be enriched with metadata information like Java annotations. This metadata can be used by the compiler, tools, and by the run-time environment to implement custom behaviours or custom handling.

It is possible to generate API documentation in a data-driven manner. By observing many programs that use a given API, it is possible to infer the typical usages, as well the required contracts and directives.Then, templates can be used to generate natural language from the mined data.