API: An **application programming interface** (**API**) is a [computing interface](https://en.wikipedia.org/wiki/Interface_(computing)) to a software component or a system, that defines how other components or systems can use it. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. It can also provide extension mechanisms so that users can extend existing functionality in various ways and to varying degrees.

A: Application

P: Programming

I: Interface. It provides the developer the means of controlling its interaction while abstracting away how that actually works, implementation-wise.

1. REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous [dissertation](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).

Like any other architectural style, REST also does have it’s own [6 guiding constraints](https://restfulapi.net/rest-architectural-constraints/) which must be satisfied if an interface needs to be referred as **RESTful**. These principles are listed below.

**Guiding Principles of REST**

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

ESSAY: