**Principles of REST:** Which an API should follow to become easy to interact with and server responds to one’s requests with resources e.g. student, chair etc. and it becomes easy to create API’s.

### Client-Server

This design principle works on the concept that client and server should be isolated from one another and permitted to develop independently. This way, you can improve manageability across numerous platforms and increase scalability by streamlining server components as user interface concerns are separate from the data storage concerns.

### Stateless

As per this principle, REST APIs are stateless, which means calls can be made independent of one another. Moreover, every call includes the data essential to complete itself effectively.

In other words, every request sent from the client to the server must include all the info needed to comprehend the request.

### Cacheable

As a stateless API can upsurge [request overhead](https://books.google.com.sa/books?id=Gg0sCgAAQBAJ&pg=PA9&lpg=PA9&dq=stateless+API+can+increase+request+overhead&source=bl&ots=IhG2ZW-Tt_&sig=ACfU3U1CLkXJeAEo9Fr8-OJUVeu8boOLog&hl=en&sa=X&ved=2ahUKEwjHrv6ssJLnAhUCC2MBHX4cDu8Q6AEwCnoECAYQAQ#v=onepage&q=stateless%20API%20can%20increase%20request%20overhead&f=false) by managing huge loads of inbound and outbound calls, a REST API design should be able to store cacheable data. According to this API design principle, data within a response should be indirectly or clearly categorized as [cacheable or non-cacheable](https://restfulapi.net/).

If a response is cacheable, the client cache is provided the right to recycle that response data for similar requests in the future.

### Uniform Interface

To decouple a client from the server, you need to have a unified interface that permits autonomous development of the application without tightly coupling its services, models, and actions to the API layer itself.

This design principle streamlines the whole system architecture and enhances the visibility of communications. To attain a uniform interface, several architectural controls are required to guide the performance of the elements within the REST API architecture.

REST is defined by four interface controls, including identification of resources, management of resources through representations, self-descriptive communications, and hypermedia as the engine of the application state.

### Layered System

REST API’s architecture includes several layers that operate together to construct a hierarchy that helps generate a more scalable and flexible application. Due to its layered system, an application has better security as components in each layer can’t interact outside the subsequent layer. Moreover, it balances loads and offers shared caches for stimulating [scalability](https://www.edureka.co/blog/what-is-rest-api/#Need%20of%20REST%20API).

A layered REST API architecture system has greater stability because it restrains component performance. so that each component can’t ‘see’ further than the immediate layer with which it is intermingling.

### Code on Demand

This design principle allows for coding or applets to be communicated through the API to be used within the application.

A REST API definition permits extending client functionality by downloading and implementing coding in the form of applets or scripts. This streamlines clients by decreasing the number of features essential to be pre-implemented.

Most of the time, a server returns static resource representation in XML or JSON format. But when required, servers can deliver executable code to the client.