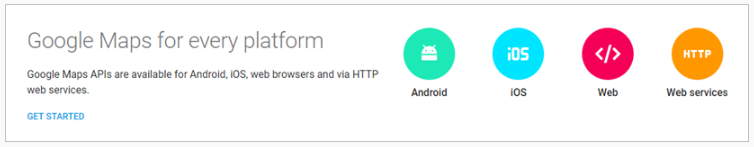
Many a times, you would want to use an already existing functionality of an application into other applications running on a different platforms and devices. For example, applications running on different platform can interact with Google Maps to provide location services.



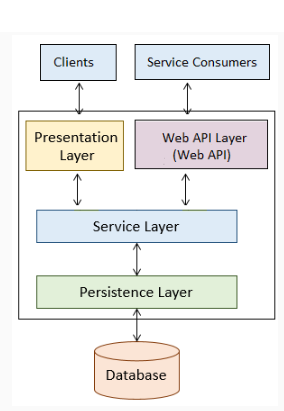
Similarly, users of Instagram, a photo sharing application, can share photographs not only with Instagram friends but also with friends on other social networking applications such as Twitter and Facebook.



Have you ever wondered how all this is possible? How application running on different platforms using different technologies can communicate with each other? All these are achieved using **Web Services**. So, let us learn what is a web service.

Note : All trademarks belong to their respective owners.

Web service is a standard way of communication between web applications running on diverse platforms and frameworks. In our scenario, Google Map is exposed as web service and consumed by different applications running on different devices. Web services are usually enclosed in the Web API layer or API layer, which communicates directly with the service layer. The Web Service layer provides the logic to convert software objects into different representations that can be sent to destination applications.



When an application is developed as a web service following are the benefits:

* Integrates with other systems easily
* Creates reusable components
* Cost savings
* Application built using different technologies can interact with each other, for example a Java based application can send data to a.NET application.

As web services provides way for communication between web application let us understand some concepts of web applications.

**Web Client**

A web client is a software using which you can interact with the server where the web application is hosted. The most commonly used web clients are web browsers such as Google Chrome, Mozilla Firefox, Internet Explorer, etc.

**Web Server**

A web server is a software where the web application is hosted. It processes the request received from the client and sends the response back to the client. It runs on some physical system and listens to client request on a specific port. Apache Tomcat is one of the most widely used web server.

**Uniform Resource Locator (URL)**

URL stands for Universal Resource Locator. It used by the web client to locate the web server and resource. Every resource on the web has its own unique URL. The following is an example of URL:

http://infy.com:8080/InfyBank/jsps/welcome.jsp

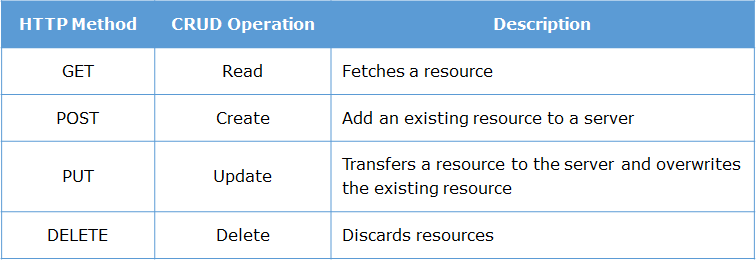
http:// – It is the communication protocol used for server-client communication.

infy.com  – It is the hostname of the server that maps to a unique IP address.

8080 – This is the port on which server is listening. It is optional and if not provided request will go to the default port of the protocol.

**Hypertext Transfer Protocol (HTTP)**

It is a protocol of communication between client and server. A client sends an HTTP request for a resource and the server returns an HTTP response with the desired resource. The HTTP request contains information about the action that the client is requesting the server to perform. This information is mentioned in HTTP request as the following methods:



**GET method**

The HTTP GET method is used to retrieve a resource. It means, “get the resource identified by this URL.”  If parameters have to be passed along with GET request then they are passed by appending a query string to the URL. For example, the following is a URL for passing tom as a username:

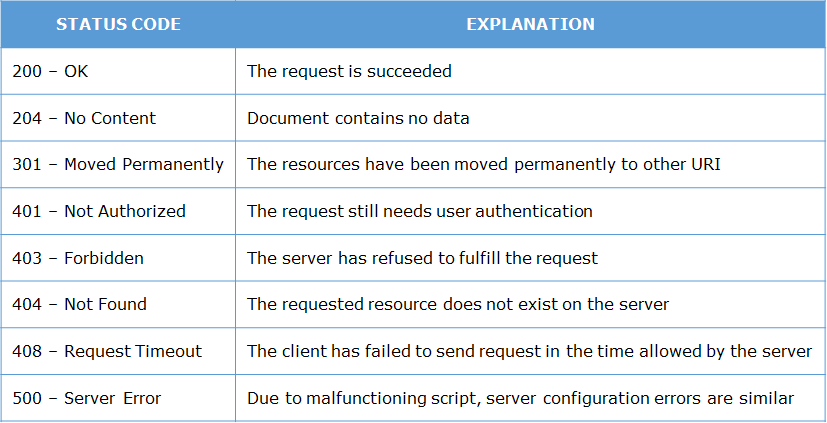
http://infy.com:8080/InfyBank?username=Tom

In above URL, the part after the question mark is called a query string. It consists of parameter name and value pairs separated by an ampersand (&). In HTTP GET request the form data gets appended to the URL which will be visible to the user

**POST method**

A POST request is used to send data to the server in order to be processed. It means,“post the data to the resource identified by this URL”. The data is sent as a body of the message.

An HTTP message sent by a server to a client is called an HTTP response. The initial line of an HTTP response is called the status line. It has three parts, separated by spaces: the HTTP version, a response status code that tells the result of the request, and an English phrase describing the status code. Some important response codes are as follows:



**Web Container**

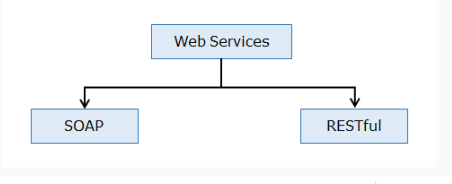
Web applications are composed of web components such as servlets, Java Server Page (JSP). These web components gets executed to generate content based on client request. So they need an environment for their execution. This execution environment is called as a web container. Web server takes the help of web container for generating dynamic content. Some commonly used web containers are

* Apache Tomcat
* GlassFish
* Jetty

**Application Server**

A web application resides in an application server. The application server provides the web application with easy and managed access to the resources of the system. It also provides low-level services, such as the HTTP protocol implementation and database connection management. A web container is just a part of an application server.

Web Services are of two types:



**RESTful Web Services**

RESTful web services are based on REST principles. As per REST principle everything is a resource.

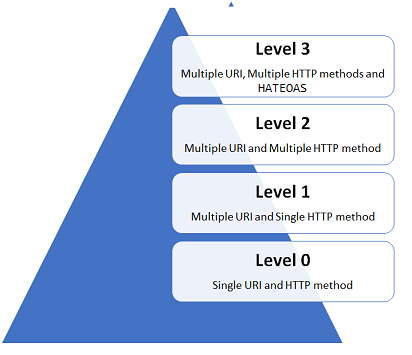
**SOAP based Web Services**

SOAP based web services are services that are described, discovered and accessed using standards recommended by W3C (not covered as part of this course).

In this course, you will be learning about RESTful Web Services.

**Richardson Maturity Model (RMM)**

The Richardson’s Maturity Model is used to divide REST based web services in different categories based on how much they follow REST principles. It was developed by Leonard Richardson. It has following levels:



**Level 0**

This is the basic level for a REST based web service. In this level single URI is used to expose the entire API. It uses a single HTTP method (typically POST or GET) to make remote procedure call on a single URI to perform operation on resources. SOAP based and XML RPC based web services comes in this level.

**Level 1**

The services in this level have multiple URI's for every resource but uses only one HTTP method – generally POST – to perform all the operations. Since specific resource is identified by a unique URI services in this level are closer to REST principles than services in level zero.

**Level 2**

The services in this level uses HTTP protocol and its methods and status codes along with URI’s to perform all the operations.  For example, to get the employees, we send a GET request with the URI /employees, and the server sends proper response 200 OK and to add a customer, we send a POST request with the URI /employees, and the server sends proper response 201 CREATED. Services which implements CRUD operations are Level 2 services.

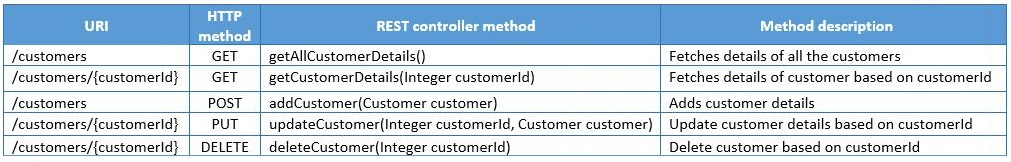
**Level 3**

This is the most mature level for a service. It is the combination of Level 2 and HATEOAS (Hypermedia as the Engine of Application State). Services in this level provides responses with links to related resources and controls which tells service client what to do next.

Suppose for InfyBank application you want to develop REST API to expose its following functionalities:

1. Add a customer
2. Get all customers
3. Update customer details
4. Fetch details of particular customer
5. Delete a customer

To do this you have to create Spring controller class and implement the REST API endpoints in it. The methods in controller class which implement REST endpoints are called as REST controller methods. Now to implement above requirements you will create CustomerAPI Spring controller class. The following table shows the REST end points, HTTP methods and REST controller methods that you are going to implement in this class:



GraphQL vs. REST

When you want to build an API, you have to use REST, which is a standard for designing web APIs. REST is used by many developers and is widely regarded as the traditional way to send data over HTTP, but this trend is changing with the increase of GraphQL popularity.

GraphQL is a new and revolutionary technology and provides an approach to develop web APIs to replace legacy REST APIs. It can be used as an alternative to REST APIs.

What is REST?

REST stands for **Representational state transfer**. It is a conventional and standard way to design web APIs. It defines a set of constraints that are used to create Web services. It was popular among developers because of its great features such as stateless servers, structured access to resources, etc. The main problem with REST is that it is not so flexible to cope up with the rapidly changing requirements of the clients that access them. That's why we need an alternative to overcome this limitation.

What is GraphQL?

So, [GraphQL](https://www.javatpoint.com/graphql) was invented to provide more flexibility and efficiency. It solves many problems and inefficiencies that developers face when working with REST APIs.

**Note:**

* We must know that GraphQL is not any type of database. It is not an alternative to SQL.
* GraphQL is also not a replacement for REST. It is only an alternative. You don't need to choose between one and the other. You can easily use both in the same project.

Differences between GraphQL and REST

|  |  |
| --- | --- |
| **GraphQL** | **REST** |
| GraphQL is invented to provide more flexibility and efficiency. It solves many problems and inefficiencies that developers face when working with REST APIs. | REST is a conventional and standard way to design web APIs. |
| GraphQL is a data query and manipulation language for APIs, a specification, and a set of tools that operates over a single endpoint using HTTP. | A REST API is an architectural concept that provides network efficiency, scalability which allows support of large numbers of components, simplicity, modifiability, visibility, portability, reliability, etc. |
| GraphQL follows client-driven architecture. | REST follows server-driven architecture. |
| GraphQL has a graphical structure. In GraphQL, objects are represented by nodes, and edges represent the relationship between nodes. | REST can be organized in terms of endpoints. |
| The development speed in GraphQL is fast. | The development speed in REST is Slow. |
| GraphQL is slightly difficult to learn. | REST is easy to learn. |
| GraphQL provides high consistency across all platforms. | In REST, it is hard to get consistency across all platforms. |
| Over the last few years, GraphQL mainly focuses on optimizing performance and flexibility. | Over the last few years, REST has mainly focused on making new APIs. |
| GraphQL is growing day by day. | REST is already an extensive community. |
| In GraphQL, error handling is difficult. | Error handling in REST is simpler as compared to GraphQL because of its rich features. |
| GraphQL doesn't have any automatic caching system. | REST uses caching automatically. |

Where GraphQL is better than REST API

When you use REST API to fetch information, it will always return a complete dataset. For example, if you want to request information from two objects, you have to perform two REST API requests. The biggest advantage of using the REST API is its simplicity. In REST APIs, you have one endpoint that does only one task, so it is easy to understand and manipulate.

In REST API, if you retrieve some information from a specific endpoint, you will always get a complete data set because, in REST APIs, you couldn't restrict the fields that the REST API returns. This is called over fetching. On the other hand, by using GraphQL, you can escape from over fetching. You can query your request to fetch exactly what you need from multiple objects. Once you get familiar with GraphQL, it is easy and powerful. It facilitates you to fetch only the required data, so; you limit the amount of processing required.

Where REST API is better than GraphQL

REST is already an industry standard for companies deploying APIs. It provides a lot of great features such as client-server architecture, stateless server, cachability, layered system, load balancer, uniform interface, etc.

# GraphQL Advantages and Disadvantages

When Facebook build GraphQL, they needed a powerful data-fetching API which can handle all the task of Facebook, yet simple and easy to learn and use by their product developers. GraphQL was developed to rebuild Facebook's native mobile applications.

GraphQL is a query language. It provides a query in the form of a string that is sent to a server. The server interprets the query and then returns the result in the form of JSON format to the client.

### Advantages of GraphQL

Following is a list of major advantages of GraphQL:

**1. GraphQL is faster**

[GraphQL](https://www.javatpoint.com/graphql) is way faster than other communication APIs because it facilitates you to cut down your request query by choosing only the specific fields you want to query.

**2. Best for complex systems and microservices**

We can integrate multiple systems behind GraphQL's API. It unifies them and hides their complexity. The GraphQL server is also used to fetch data from the existing systems and package it up in the GraphQL response format. This is most beneficial for legacy infrastructures or third-party APIs that are enormous in size and difficult to maintain and handle.

When we have to migrate from a monolithic backend application to a microservice architecture, the GraphQL API can help us to handle communication between multiple microservices by merging them into one GraphQL schema.

**3. No over-fetching and under-fetching problems**

The main advantage of GraphQl over REST is that REST responses contain too much data or sometimes not enough data, which creates the need for another request. GraphQL solves this problem by fetching only the exact and specific data in a single request.

**4. Hierarchical Structure**

GraphQL follows a hierarchical structure where relationships between objects are defined in a graphical structure. Here, every object type represents a component, and every relationship field from an object type to another object type represents a component wrapping another component.

**5. Defines a data shape**

When we request GraphQL queries to the server, the server returns the response in a simple, secure, and predictable shape. So, it facilitates you to write a specific query according to your requirement. This makes GraphQL really easy to learn and use.

**6. Code-sharing**

We can share the GraphQL fields used in multiple queries at a higher component level for reuse. This feature is referred to as fragments and allows you to get different data while keeping the same schema field.

**7. Strongly typed**

GraphQL is a strongly typed language where each level of a GraphQL query corresponds to a particular type, and each type describes a set of available fields. So, it is similar to SQL and provides descriptive error messages before executing a query.

**8. Protocol, not a storage**

Arbitrary functions back GraphQL fields on the servers. They do not dictate or provide any backing storage. Instead, GraphQL takes advantage of your existing code.

**9. Introspective**

We can query a GraphQL server for its supporting types. It creates a powerful platform for tools and client software such as application framework, Relay, or IDEs like GraphiQL. GraphiQL facilitates developers to learn and explore an API quickly.

**10. Latest version not required**

In GraphQL, the result set or returned data is very specific according to the client's query, so; it is very simple and easy for the server to generalize it. When we add new product features, additional fields to the server, they don't affect the existing clients. You can use the older server without any worry because server fields can be deprecated but continue to function. This compatible process doesn't require the need for an incrementing version number. You can see that Facebook is using the same version of GraphQL API in their applications.

### Disadvantages of GraphQL

Although GraphQL has negligible disadvantages over its advantages, we are giving some disadvantages here. Following is the list of disadvantages of GraphQL:

**1. GraphQL Query Complexity**

Don't mistake GraphQL as a replacement for server-side databases. It is just a simple query language. When a query is requested, the server performs database access. When we have to access multiple fields in one query whether it is requested in a RESTful architecture or GraphQL, the varied resources and fields still have to be retrieved from a data source. So, it also shows the same problems when a client requests too many nested fields data at a single time. So there must be a mechanism like maximum query depths, query complexity weighting, avoiding recursion, or persistent queries to stop inefficient requests from the client-side.

**2. GraphQL Caching**

It is more complicated to implement a simplified cache with GraphQL than implementing it in REST. In REST API, we access resources with URLs, so we can cache on a resource level because we have the resource [URL](https://www.javatpoint.com/url-full-form) as an identifier. On the other hand, In GraphQL, it is very complex because each query can be different, even though it operates on the same entity. But most of the libraries built on top of GraphQL offer an efficient caching mechanism.

**3. GraphQL Rate Limiting**

Another problem with GraphQL is rate-limiting. In REST API, you can simply specify that we allow only this amount of requests in one day", but in GraphQL, it is difficult to specify this type of statement.