**Web service:**

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system.

XML is used to encode all communications to a web service. For example; a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with UNIX applications. Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents. tier

**Three tier Architecture:**

Three-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture.

The three tiers in three-tier architecture are:

1. Presentation Tier: Occupies the top level and displays information related to services available on a website.

This tier communicates with other tiers by sending results to the browser and other tiers in the network.

2. Application Tier: Also called the middle tier, logic tier, business logic or logic tier, this tier is pulled from the presentation tier.

It controls application functionality by performing detailed processing.

3.Data Tier: Houses database servers where information is stored and retrieved.

Data in this tier is kept independent of application servers or business logic.

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability.

**Microservices:**

The central idea behind microservices is that some types of applications become easier to build and maintain when they are broken down into smaller, composable pieces which work together. Each component is continuously developed and separately maintained, and the application is then simply the sum of its constituent components. This is in contrast to a traditional, "monolithic" application which is all developed all in one piece.

**Single-page application vs. multiple-page application:**

Single-Page Application

A single-page application is an app that works inside a browser and does not require page reloading during use. You are using this type of applications every day. These are, for instance: Gmail, Google Maps, Facebook or GitHub. SPAs are all about serving an outstanding UX by trying to imitate a “natural” environment in the browser — no page reloads, no extra wait time. It is just one web page that you visit which then loads all other content using JavaScript — which they heavily depend on.

SPA requests the markup and data independently and renders pages straight in the browser. We can do this thanks to the advanced JavaScript frameworks like AngularJS, Ember.js, Meteor.js, Knockout.js . Single-page sites help keep the user in one, comfortable web space where content is presented to the user in a simple, easy and workable fashion.

Multi-Page Application

Multiple-page applications work in a “traditional” way. Every change eg. display the data or submit data back to server requests rendering a new page from the server in the browser. These applications are large, bigger than SPAs because they need to be. Due to the amount of content, these applications have many levels of UI. Luckily, it’s not a problem anymore. Thanks to AJAX, we don’t have to worry that big and complex applications have to transfer a lot of data between server and browser. That solution improves and it allows to refresh only particular parts of the application. On the other hand, it adds more complexity and it is more difficult to develop than a single-page application.

**Rest Api:**

A restful api is an API that uses HTTP request to get, put, post and delete data. Developer will set the requirements and responses of the api. Each URL is a request and data sent back to client called response.

**Factory pattern:**

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

**Singleton pattern:**

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

**Prototype pattern:**

Refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

**Composite pattern:**

Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern comes under structural pattern as this pattern creates a tree structure of group of objects.

This pattern creates a class that contains group of its own objects. This class provides ways to modify its group of same objects. We are demonstrating use of composite pattern via following example in which we will show employees hierarchy of an organization.