**DAY 1**

**SDLC**

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software’s. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software.

**WaterFall**

Waterfall methodology is a linear project management approach, where stakeholder and customer requirements are gathered at the beginning of the project, and then a sequential project plan is created to accommodate those requirements. There are five phases of waterfall method Requirements, Design, Implementation, Verification, Maintenance. Advantages of the method are it is easy to understand, fewer production issues, better budget management.

**AGILE**

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages more frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one. The process is lightweight Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one. It breaks down the agile method into several smaller incremental releases of the products called sprints, until the product is complete.

**Scrum Master**

The name was initially intended to indicate someone who is an expert at Scrum and can therefore coach others. The Scrum Master is also responsible for improving interactions between the Scrum team and the organization to maximize the productivity of the Scrum team. Finally, the Scrum Master is responsible to arranges and facilitates the team’s meetings.

**Product backlog**

The product backlog is compiled of all the things that must be done to complete the whole project. But it’s not just a simple list. An effective product backlog breaks down each of the items on the list into a series of steps that helps the development team.

Sprint Backlog is a subset of Product Backlog it contains only that item, or those items, that can be completed during each sprint.

Velocity

**Velocity** is a measure of the amount of work a Team can tackle during a single Sprint and is the key metric in **Scrum**. **Velocity** is calculated at the end of the Sprint by totaling the Points for all fully completed User Stories.

**Epic**

Are large bodies of work that can be broken down into a number of smaller tasks, Epics are a helpful way to organize your work and to create a hierarchy. Stories also called “user stories,” are short requirements or requests written from the perspective of an end user.

A sprint employs four different scrum ceremonies to ensure proper execution: sprint planning, daily scrum, sprint review and sprint retrospective.

1) Sprint Planning: This is where the team meets and decides what they need to complete in the coming sprint

2) Daily Scrum: This is a standup meeting, or a very short – 15-minute mini-meeting – for the team to make sure they’re all on the same page.

3) Sprint Review: This is another type of meeting, but one in which the team demos what they shipped in the sprint.

4) Sprint Retrospective: This is when the team reviews their work, identifying what they did well and what didn’t go as planned, so they can make the next sprint better.

**Grooming**

Grooming is when the product owner and some, or all, of the rest of the team review items on the backlog to ensure the backlog contains the appropriate items, that they are prioritized, and that the items at the top of the backlog are ready for delivery. This activity occurs on a regular basis and may be an officially scheduled meeting or an ongoing activity.

**How is Jira Usefull**

Jira Scrum board is designed so teams can organize their work around the Sprint timeframe. Jira Scrum Boards provide transparency into the team's work by slicing work into stages and utilizing burndown and velocity reports.

**Waterfall and SCRUM**

The waterfall development model or traditional software development life cycle. Its working approach is linear and sequential, it completes one activity before starting the other activity. Waterfall’s working style break up the work into the requirement, analysis, design, coding and testing, and term that phases. This works well with smaller projects

This is a member of the agile family. Scrum puts the focus on the management and development of the project. Scrum process is used to manage, develop and deliver the project on time. Scrum works best for complex projects and innovative solutions are delivered.

**Product Owner Responsibilities**

Defining the vision, Managing the product backlog, Prioritizing needs, Overseeing development stages, Anticipating client needs, Evaluating product progress at each iteration.

**Day 2**

**World Wide Web**

A Web server is software or hardware that uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests made over the World Wide Web (WWW).

**Characteristics of a client server computing.**

The client server computing works with a system of request and response. The client sends a request to the server and the server responds with the desired information.

The client and server should follow a common communication protocol so they can easily interact with each other. All the communication protocols are available at the application layer.

A server can only accommodate a limited number of client requests at a time. So it uses a system based to priority to respond to the requests.

Denial of Service attacks hinder a servers ability to respond to authentic client requests by inundating it with false requests.

**Presentation layer** is concerned with the representation of information as data, A major function of this is also making sure that data going in can be used by the local node, and that data going out can be used by the remote node.

A **database server** is a server which uses a database application that provides database services to other computer programs or to computers.

A **super-server** starts other servers when needed, normally with access to them checked by a TCP wrapper. It uses very few resources when in idle state. This can be ideal for workstations used for local web development, client/server development or low-traffic daemons with occasional usage.

A **2-tier** architecture is a software architecture in which a presentation layer or interface runs on a client, and a data layer or data structure gets stored on a server.

A **3-tier** architecture is a type of software architecture which is composed of three “tiers” or “layers” of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers.

**File Server** The term server highlights the role of the machine in the traditional client–server scheme, where the clients are the workstations using the storage. A file server does not normally perform computational tasks or run programs on behalf of its client workstations.

**SOA & MicroServices**

Service-oriented architecture (SOA) enables increased business agility, improved business workflows, extensible architecture, enhanced reuse, and a longer life span of applications.

**reusability Service** is a design principle, applied within the service-orientation design paradigm, to create services that can be reused across a business. These reusable services are designed so that their solution logic is independent of any business process or technology.

**Differences between Web services and SOA**. Web services define a web technology that can be used to build applications that can send /receive messages using SOPA over HTTP. However, SOA is an architectural model for implementing loosely coupled service based applications. Web services can be used to implement SOA applications.

**Disadvantages of SOA**

High Bandwidth Server – As therefore net service sends and receives messages and knowledge often times so it simply reaches high requests per day. So it involves a high-speed server with plenty of information measure to run an internet service.

Extra Overload – In SOA, all inputs square measures its validity before it’s sent to the service. If you are victimization multiple services then it’ll overload your system with further computation.

High Cost – It is expensive in terms of human resource, development, and technology.

An **enterprise service bus** (**ESB** for short) refers to software architecture that allows for the integration of enterprise applications and services, such as middleware infrastructure platforms.

No in **SOA** we don’t need build a system from scratch, if we need to integrate any existing system you just can loosely couple wrappers which help in wrapping all customer services and expose all functionalities in a generic manner.

**Cultural**. SOA does require people to think of business and technology differently. Instead of thinking of technology first (e.g., If we implement this system, what kinds of things can we do with it?), practitioners must first think in terms of business functions, or services (e.g., My company does these business functions, so how can I set up my IT system to do those things for me most efficiently?).It is expected that adoption of SOA will change business IT departments, creating service-oriented (instead of technology-oriented) IT organizations.

Advantages of **microservice architecture**

* When a business offers features that are clearly separated from each other, this architectural pattern can make their app highly scalable.
* Individual services may have different demand profiles, therefore, the business will implement scaling strategies for these individual services. This helps with optimizing and prioritizing resources.
* You will find it much easier to read and understand your codebase if you use this architecture pattern.
* It’s easier to maintain the app.
* Individual microservices can be deployed separately. You deploy only the microservice that you have changed and not the entire application. This reduces the time and effort spent on your deployment process.
* You find it easier to debug your app since you don’t need to review multiple layers of a giant app.
* The microservices architecture pattern makes it easier to isolate faults.
* You build more resilient services if you use the microservices architectural pattern, and this improves the fault tolerance of your app.
* You improve reusability if you use the microservices architecture. In this patter, you build and organize microservices around business functionalities. When there are commonalities with other business functions, you can reuse the microservice you developed earlier with minimal changes. Such reuse helps to reduce your development costs.

Best Practices for microservices Are Create a Separate Data Store for Each Microservice ,Keep Code at a Similar Level of Maturity, Do a Separate Build for Each Microservice, Deploy in Containers, Treat Servers as Stateless

**Monolithic**, in this context, means composed all in one piece. Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled. **SOA** is a modular means of breaking up monolithic applications into smaller components, while **microservices** provides a smaller, more fine-grained approach to accomplishing the same objective.

Q Challenges:

**Managing Microservices**

As the number of microservices increases, managing them gets more challenging. It is important that management is planned before or while microservices are being built. While the modularity helps, things can very quickly get out of hand if not managed well.

**Monitoring**

The traditional forms of monitoring and diagnostics will not align well with microservices since you have multiple services making up the same functionality previously supported by a single application. When a problem arises in the application, finding the root cause can be challenging if you do not have a means of monitoring and tracking the path a specific request took, like how many and which microservices were traversed for a specific request coming from a user interface.

**Embracing DevOps Culture**

Separate teams need agility, autonomy, and continuous delivery to be able to deliver initial releases and subsequent iterative changes. A lack of DevOps culture can bottle up releases and impact the overall time to market and the response to business requests and issues.

**Fault Tolerance**

It is important that individual services do not bring down the overall system. Fault tolerance at the service level, and more importantly, at the overall solution level, is critical. Given the complexity of a microservices environment and the complex dependency chains, failure is inevitable.

**Testing**

Testing is much more complex in a microservices environment due to the different services, their integration, and interdependencies. The team members responsible for quality assurance need to be knowledgeable on the order and channels of communications between services to have full coverage in their test cases.

The Basic Features of HTTP (Hyper Text Transfer Protocol):

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

HTTP is media independent: It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.

HTTP is connectionless: It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.

HTTP is stateless: The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

**HTTP defines a set of request methods** to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs.

Difference between **GET and POST** method in HTTP. Both GET and POST method is used to transfer data from client to server in HTTP protocol but Main difference between POST and GET method is that GET carries request parameter appended in URL string while POST carries request parameter in message body which makes it more secure way of transferring data from client to server in http protocol.

HTTP response **status codes** indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

Informational responses (100–199),

Successful responses (200–299),

Redirects (300–399),

Client errors (400–499),

and Server errors (500–599).

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

General-header: These header fields have general applicability for both request and response messages.

Client Request-header: These header fields have applicability only for request messages.

Server Response-header: These header fields have applicability only for response messages.

Entity-header: These header fields define meta information about the entity-body or, if no body is present, about the resource identified by the request.

A Uniform Resource Identifier is a string of characters that unambiguously identifies a particular resource. To guarantee uniformity, all URIs follow a predefined set of syntax rules, but also maintain extensibility through a separately defined hierarchical naming scheme.

An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over.

HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server. HTTP messages are composed of textual information encoded in ASCII, and span over multiple lines.

**Session state**, in the context of .NET, is a method keep track of the a user session during a series of HTTP requests. Session state allows a developer to store data about a user as he/she navigates through ASP.NET web pages in a .NET web application.

Representational state transfer (REST) is a style of software architecture. As described in a dissertation by Roy Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web. RESTful is typically used to refer to web services implementing such an architecture.

The following subset of HTTP methods are supported for the REST BC:

GET

The GET method retrieves specific information from the server as identified by the request URI.

PUT

The PUT method requests that the message body sent with the request be stored under the location provided in the HTTP message.

DELETE

The DELETE method deletes the specified resources.

POST

The POST method modifies data on the server from which a request was sent.

HEAD

The HEAD method is similar to the GET method except the message body is not returned in the response. The response only includes metainformation, such as a response code or corresponding headers.

**RESTful web service**s are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

**A resource** is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it. It is similar to an object instance in an object-oriented programming language, with the important difference that only a few standard methods are defined for the resource (corresponding to the standard HTTP GET, POST, PUT and DELETE methods), while an object instance typically has many methods.

REST is web standards based architecture and uses **HTTP** **Protocol**. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

**RESTfu**l **Web Services** - **Message**s. Advertisements. RESTful Web Services make use of HTTP protocols as a medium of communication between client and server. A client sends a message in form of a HTTP Request and the server responds in the form of an HTTP Response.

When a **caching mechanism** is in place, it helps improve delivery speed by storing a copy of the asset you requested and later accessing the cached copy instead of the original.

The key differences. **SOAP** is a protocol. **REST** is an architectural style. An API is designed to expose certain aspects of an application's business logic on a server, and SOAP uses a service interface to do this while REST uses URIs.

Advantages Of Stateless

As the server does not need to manage any session, deploying the services to any number of servers is possible, and so scalability will never be a problem

No states equals less complexity; no session (state) synchronize logic to handle at the server side

As the service calls (requests) can be cached by the underlying application, the statelessness constraint brings down the server's response time, that is, it improves performance with regard to response time.

**Disadvantage**

increase per-interaction overhead.

Each request of webservices needs to get additional information so that it get parsed (interpreted) so that the server understands the client state from the incoming request and takes care of the client / server sessions if needed

A **class** is a blueprint from which you can create the instance, i.e., objects. An object is the instance of the class, which helps programmers to use variables and methods from inside the class. A class is used to bind data as well as methods together as a single unit. object acts as a variable of the class.