**MicroServices** (25th Oct 2019)

=================================PolyGlot=============================================

**What is Monolithic Architecture?**

All application modules are part of single project

Features:

Old and easy approaches

No headache of implementing communication

Easy to deploy and develop

Tightly coupled

Choice of languages and frameworks are less

Every time we need to redeploy the app when a change comes in any on module

Scaling of individual modules not possible

Limited/complex polyglot persistence

**What is Service Oriented Architecture?**

Create each module of app is created as individually deployable projects

We use a communication channel which is used by all the modules to exchange the data

ESB – Enterprise Service Bus (ESB Mule)

All modules shares a common database

**What is Micro services?**

Individual developed, tested and deployed

They never share the database

Any service can communicate to any other service directly

Every service is exposing a well-defined end point – REST endpoint

Choice of languages and frameworks

Database not shared

Independently developed, tested and deployed

Independently scalable services

Communication pattern:

HTTP based – REST

Message based – Queues

**Micro services Communication Patterns:**

Rest based (HTTP)

Nearly real time

Synchronous in nature

One to one communication

For every request there is a response received

Queue based (MSMQ)

Not real time

Asynchronous in nature

One to many and one to one are supported

No response required for message based

Event based

Combination of Rest and Queue

Nearly real time

Asynchronous in nature

One way communication

No response received

One or more event handler can catch the event data

.NET Core Web API:

Every controller is inheriting from ControllerBase class

ControllerBase is a base class for MVC controller and Web API controller

Every controller is decorated with [ApiController] attribute

No RoutePrefix for controller, Route() is used instead for defining prefix.

No Route() attribute required for action, http method selector attribute can define the template and route name.

Method return type:

Void

Specific types (string, bool, integer etc.)

HttpResponseMessage

ActionResult<T> - recommended

**CORS Policies: [EnableCors(“<CorsName>”)]**

Cross origin resource sharing.

Defines a set of rules that tells the API to allow certain domain, methods and headers.

services.AddCors(cors =>

{

// Default Policy

//cors.AddDefaultPolicy(x => x.AllowAnyOrigin()

//.AllowAnyMethod()

//.AllowAnyHeader());

// Custom Policy 1

cors.AddPolicy("AllowPartners", x =>

{

x.WithOrigins("http://microsoft.com", "https://synegrtics.com")

.WithMethods("GET", "POST")

.AllowAnyHeader();

});

// Custom Policy 2

cors.AddPolicy("AllowAll", x =>

{

x.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();

});

});

Top of controller add this attribute to apply Cors: [EnableCors("AllowPartners")]

Add attribute for get multiple status code as a return

[ProducesResponseType((int)HttpStatusCode.OK)]

[ProducesResponseType((int)HttpStatusCode.NotFound)]

**Open API Documentation:**

For web services we get service documentation using WSDL file.

For RESTful services we can use **Open API specification** doc (Swagger doc)

We can use **Swashbucle.AspNetCore** package to configure the Swagger Doc.

services.AddSwaggerGen(options =>

{

options.SwaggerDoc("v1", new Info

{

Title = "Catalog API",

Description = "Catalog management API methods for eShop application",

Version = "1.0",

Contact = new Contact

{

Name = "Amol Patole",

Email = "amolpatole9000@gmail.com",

Url = "http://google.com"

}

});

});

// To show Swagger UI

if (env.IsDevelopment())

{

app.UseSwaggerUI(config =>

{

config.SwaggerEndpoint("/swagger/v1/swagger.json", "Identity API");

config.RoutePrefix = "";

});

}

**Enable Xml serializer:**

Use the AddXmlDataContractSerializerFormatters() method with service.AddMvc()

**Authentication for web API:**

No login GUI is provided for APIs

Use tokens for authentication (OAuth Token, JWT Token)

A token is an encrypted text that contains the audience (API), issuer (identity provider),

Claims (roles and other info), expiry time, algorithm and secret code

**Custom Formatters:**

**RabbitMQ:**

Communication patterns:

RESTful services – Synchrnous

Message Based – Asynchronous, one to many, not real time

Event driven architecture - Asynchronous, one to many, real time

**Messaging:**

Two application can communicate asynchronously.

A small text message is sent to a message broker.

A message broker is an agent that provides message based communication.

Features: Provides storage for messages.

Different message delivery patterns or exchange type

Direct – one to one

Topic – pattern based (multicast means in group)

Fanout – broadcasting to everyone

Deliver message when recipient comes online.

TTL – Time to Live – undelivered messages can be moved to Dead letter queue.

**Azure Service Bus, Azure Storage Queue, RabbitMQ etc.**