```
============
Webservices
============

-> It is a distributed technology which is used to develop distributed application.

-> If one application is communicating with another application then those apps are called as
distributed applications.

-> Distributed applications should have intereoperability

-> Intereoperability means language independent & platform independent.

                                java app <--------> python app

                                java app <--------> dot net app

                                java app <---------> php app

                                java app <---------> Angular app

                                java app <---------> React app


Note: Distributed applications are used for code reusability

          (Business to business communication)


=> Distributed applications can be developed in 2 ways

                        1) SOAP Webservices (Outdated)

                        2) RESTFul Services (trending)

====================
What is REST API ?
====================

 Distributed App = rest api / restful service / restful webservice/microservice


=> REST API means an application which provides business logic to other applications through
internet.

======================
REST API Architecture
======================

1) Provider / Resource

2) Consumer / Client

=> Provider means the application which is providing business services to other applications.

=> Consumer means the application which is accessing business services from other applications.

                                Ex: MakeMyTrip & IRCTC

Note: We will use JSON to exchange data between provider & consumer.

=====
Task
=====

1) What is HTTP
2) HTTP methods & Purpose of those methods
3) Http Status Codes
```

```
=================
What is HTTP
=================

=> HTTP stands for Hyper Text Transfer Protocol

=> HTTP acts as mediator between Client & Server

=> HTTP is stateless protocol

=> We need to know about

                1) Http Request

                2) Http Response

                3) Http Methods

                4) Http Status Codes


========================
HTTP Request Structure
========================

=> It contains below parts

                a) Request Line (Http Method + URL)
                b) Request Headers (Metadata) (K-V)
                c) Request Body (Payload - text/xml/json)

============================
HTTP Response Structure
============================

=> It contains below parts

                a) Response Line (Status Code + Status MSG)
                b) Response headers (Metadata - K & V)
                c) Response body (Payload - text/xml/json)

==============
HTTP Methods
==============

GET  ---> It is used to get the data (no request body)

POST ---> Send data (create record)

PUT  ---> Update record (complete update)

PATCH ---> Partial Update

DELETE ---> Delete Record

===================
HTTP Status Codes
===================

1xx (100-199) : Informational

2xx (200-299) : Success

3xx (300-399) : Redirectional

4xx (400-499) : Client Error

5xx (500-599) : Server Error
```

```
404 - Resource Not Found

500 - Internal Server Error

200 - OK

=====================================
Developing REST APIs using Spring Boot
=====================================

=> We will use 'spring-boot-starter-web' dependency

=> As part of REST API development we need to learn both

                1) Provider Development
                2) Consumer Development

=> We will use below annotations as part of Provider development

@RestController (@Controller + @ResponseBody)

@GetMapping
@PostMapping
@PutMapping
@PatchMapping
@DeleteMapping

@RequestParam
@PathVariable

@RequestBody

=> We have below 3 options to develop Consumer

                        1) RestTemplate --- sync
                        2) WebClient (webflux) --- sync & async
                        3) FeignClient (spring cloud)


Note : We will use POSTMAN to test provider functionality.

Note: We will use Swagger to generate provider documentation.

=================
HTTP GET Request
=================

=> GET request is used to get data from Server/Provider.

=> GET request will not contain request body.

=> If we want to send data in GET request then we need to use

                1) Query Parameters
                2) Path Parameters

        Ex-1 : http://localhost:8081/welcome?name=raju

        Ex-2: : http://localhost:8081/greet/john

=> Query Parameters we can read using @RequestParam annotation

=> Path Parameters we can read using @PathVariable annotation.

=====================================================================
@RestController
public class DemoRestController {
```

```
        @GetMapping(value = "/msg", produces = "text/plain")
        public ResponseEntity<String> getMsg() {
                String msg = "Welcome to Ashok IT";
                return new ResponseEntity<>(msg, HttpStatus.OK);
        }

        @GetMapping("/greet/{name}")
        public String getGreetMsg(@PathVariable("name") String name) {
                String msg = name + ", Good Morning..!!";
                return msg;
        }

        @GetMapping("/welcome")
        public String getWelcomeMsg(    ") String name) {
                String msg = name + ", Welcome to REST API";
                return msg;
        }
}
=========================================================================
@RestController
public class CustomerRestController {

        @GetMapping(value = "/customers", produces = "application/json")
        public ResponseEntity<List<Customer>> getCustomers() {
                Customer c1 = new Customer(1, "John", "john@gmail.com");
                Customer c2 = new Customer(2, "Smith", "smith@gmail.com");
                Customer c3 = new Customer(3, "David", "david@gmail.com");
                List<Customer> list = Arrays.asList(c1, c2, c3);
                return new ResponseEntity<>(list, HttpStatus.OK);
        }

        @GetMapping(value = "/customer", produces = "application/json")
        public ResponseEntity<Customer> getCustomer() {
                Customer c = new Customer(1, "John", "john@gmail.com");
                return new ResponseEntity<>(c, HttpStatus.OK);
        }
}
============================================================

==================
HTTP Post Request
==================

=> POST method is used for creating new resource/record

=> POST method contains request body

==================
HTTP PUT Request
==================

=> PUT method is used for updating resource/record

=> PUT method contains request body

==================
HTTP DELETE Request
==================

=> DELETE method is used for deleting resource/record

=> DELETE method contains request body


produces : Represents rest api method response body data format

consumes : Represents rest api method request body data format
```

```
@RequestParam : To read query params from url

@PathVariable : To read path params from URL

@RequestBody : To read data from http req body (payload)

==================
What is Swagger ?
==================

=> Swagger is used to generate documentation for REST API

=> Using Swagger we can test rest api.

=> Add below dependency in pom.xml file

<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.5.0</version>
</dependency>

=> After running the application use below url to access swaager-documentation

               URL : http://localhost:8081/swagger-ui/index.html/

=========================================================================
Assignment -1 : Develop Spring Boot REST API to perform Crud operations with Database table using
Spring Data jpa and test it using Swagger documentation.

@@ Referenece Video : https://www.youtube.com/watch?v=_rOUDhCE-x4
=========================================================================

Assignment - 2 : Develop Ticket Booking Rest api (IRCTC api) with below operations. Use H2 DB for
saving data.

        operation-1 : book-ticket
        operation-2 : get-ticket using ticket-id
        operation-3 : get-all-tickets

input : passenger-data
                          - name
                          - dob
                          - gender
                          - doj
                          - from
                          - to
                          - trainNum

output : ticket-data
                          - ticketId
                          - ticketStatus
                          - trainNum
                          - from
                          - to
                          - doj
                          - name

-------------------------------------------------------------------------


===================
What is xml ?
===================

-> XML stands for extensible markup language

-> xml is free and open source
```

-> xml govenred by w3c org

-> xml represents data in the form of elements

-> XML is intereoperable (language independent)

-> XML is used to represent the data

---
syntax:

```
<person>
        <id>101</id>
        <name>Ashok</name>
</person>
```

-> We have 2 types of elements in xml

1) simple element (contains data directley)

2) compound element (contains child elements)

==========
Dealing with xml data in java applications
==========

=> Upto java 8v we have JAX-B api in jdk to deal with xml files in java.

=> Using JAX-B api we can convert java object to xml and vice versa.

```
                                        jax-b
                    java obj <---------> xml data
```

Note: From java 9 onwards jax-b api is not part of JDK software.

=> If we want to deal with xml data in java applications, we need to add dependency

Marshalling : Converting java object to xml data

Un-Marshalling : Converting xml data to java object

=> To deal with xml data in spring boot rest api we need to add below dependency in pom.xml file

```
<dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

=> Below is the rest controller method which supports both xml and json response

```
@RestController
public class ProductRestController {

        @GetMapping(
                        value = "/product",
                        produces  =  {"application/xml" , "application/json"}
        )
        public ResponseEntity<Product> getProduct() {
                Product p = new Product(101, "Monitor", 1000.00);
                return new ResponseEntity<>(p, HttpStatus.OK);
        }
}
```

Note: The above method returning Object that means it is loosely coupled with xml and json

formats.

=> Consumer can decide in which format they want response.

=> Consumer should send Accept header in the request to decide response format.

Accept = appliction/json

Accept = application/xml

Note: Based on the Accept header value given by consumer response will be sent from the provider.

==========
what is Content-Type header ?
==========

=> It is used to represent in which format consumer sending request body data to provider.

```
@PostMapping(
                value="/product",
                consumes = {"application/xml", "application/json"},
                produces = "text/plain"
)
public ResponseEntity<String> addProduct(@RequestBody Product p){
        System.out.println(p);
        //logic
        return new ResponseEntity<String>("product added", HttpStatus.CREATED);
}
```

=> Http Headers

Accept : In which format client expecting response body
Content-Type : In which format client giving request body

=> Media Types

consumes : In which format provider expecting input
produces : In which format provider can give output

==========

1) What is Distributed Application & Why ?

2) What is Intereoperability

3) JSON & Jackson

4) XML & JAX-B (outdated)

5) HTTP Protocol (mediator)

6) HTTP Methods (GET, POST, PUT, PATCH, DELETE)

7) HTTP Status Codes

8) HTTP Request Structure

- Request Line (Method + URL)
- Request Headers (Accept, Content-Type)
- Request Body (payload - txt/json/xml)

9) HTTP Response Structure

- Response Line (Status Code + MSG)
- Response Headers (meta data)

- Response Body (payload - txt/json/xml)

10) REST API Architecture

- Provider
- Consumer

11) @RestController

12) ResponseEntity

13) @RequestParam (Query Params)

14) @PathVariable (Path params)

15) @RequestBody

16) Consumes & Produces (Media Types)

17) Accept & Content-Type headers

18) POSTMAN

19) Swagger

20) IRCTC API

21) CRUD Operations using REST API with Data JPA

=========
Consumer
==========

=> The application which is accessing other applications is called as Consumer application.

=> Using Spring Boot we can develop Consumer in 3 ways

1) RestTemplate (outdated)

2) WebClient

3) FeignClient

=> RestTemplate supports only Synchronus communication

=> WebClient supports both sync & async communication (spring 5.x)

=> Feign Client is used for inter service communication in Microservices

==================================
What is Synchronus Communication ?
==================================

=> After sending request to provider if consumer is waiting for the response then it is called as Synchronus communication.

==================================
What is Asynchronus Communication ?
==================================

=> After sending request to provider if consumer is not waiting for the response then it is called as asynchronus communication.

================
Third Party API
================

URL : https://api.restful-api.dev/objects/

GET Request

input : N/A

output : application/json

==========
Assignment
==========

=> Develop MakeMyTrip application to book train tickets by communicating wit IRCTC API.

Screen-1 : book-ticket page

Screen-2 : get tickets

1) Class Based Exception Handling

2) Gloabl Exception Handling

@ControllerAdvice - Class level (C 2 B)

@ExceptionHandler - Method level

@RestControllerAdvice - Class Level

==========
Can we develop REST API without creating
@Controller and @RestController class ?
==========

Spring Data REST - @RestRepositories

=> In Spring Boot, We can develop REST API in 2 ways

        1) Spring Web MVC ----> @RestController

        2) Spring Data REST ---> @RestRepository

```
@RepositoryRestResource(path = "books")
public interface BookRepository extends JpaRepository<Book, Integer>{

}
```

Q) How to to restrict PUT and DELETE requests in Rest repository ?
