# REST API Development Material

-> Webservices is a distributed technology which is used to develop distributed applications.

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

Ex- :

www.ashokit.in -------> Whatsapp / RazorPay / Teachable

Gpay ----------> SBI Bank App

MakeMyTrip ------> IRCTC App

frontend app -------> Backend App

(angular) (Java SB)

=> Distributed applications are used for Business 2 Business Communication (B 2 B).

-> Distributed applications should have interoperability.

-> Interoperability 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

=> Web Services we can develop in 2 ways

1) SOAP Web services (xml based and out-dated)

2) Restful web services (Trending)

## What is REST API ?

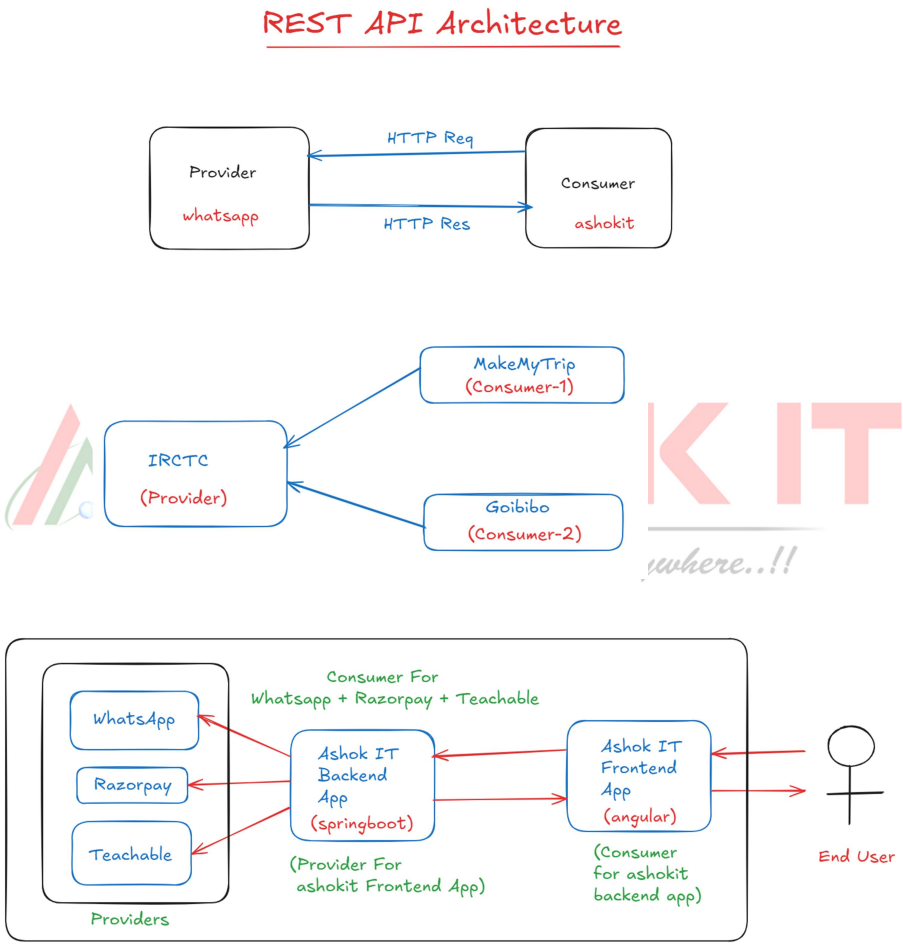
=> REST API means one distributed application which will provide services to other applications.

REST API = REST Service / Restful Web services / Micro services

# REST API Architecture

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

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




**Note: Consumer and Provider applications will exchange data using JSON format.**

# What is JSON

**=> JSON stands for Java Script Object Notation.**

**=> JSON represents data in key-value format.**

**Ex :**

```
{
    "id": 101,
    "name": "Ashok",
    "phno": 979799,
    "gender": "Male"
}
```

**Note: Key is a string so we will keep it in double quotes. If value is also a string then keep it in double quotes.**

**=> JSON is very light weight.**

**=> JSON is platform independent & language independent.**

**=> JSON is interoperable.**

**=> JSON is used to transfer data over a network.**

**=> Distributed applications will use JSON data for request & response.**

**Ex : MakeMyTrip <---------- json ----------> IRCTC**

## Working with JSON in Java Applications

**=> In java applications we can represent data in json format.**

**=> To work with JSON in Java app, we have third party libraries**

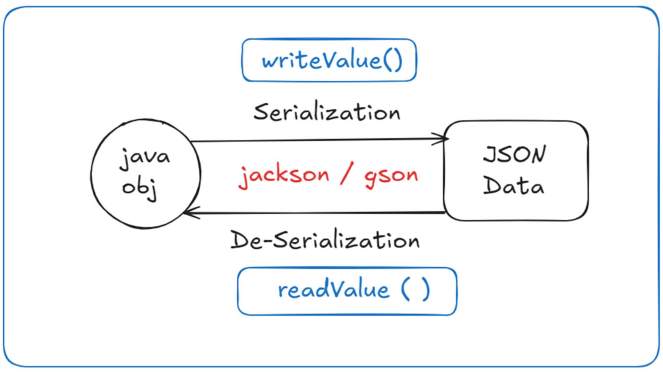
**a) Jackson**

**b) Gson (google)**

**=> By using above libraries we can convert JSON data to Java Object and vice versa.**

**=> The process of converting java obj to json is called as Serialization.**

**=> The process of converting json data to java obj is called as de-serialization.**




## Jackson API

**1) Create Maven Project using IDE**

**2) Add below dependency in pom.xml file**

```
    <dependencies>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.17.0</version>
        </dependency>
    </dependencies>
```

**3) Create binding class to represent data**

**public class Customer {**

**private Integer id;**

**private String name;**

**private Long phno;**

**// setters & getters**

**// toString ( )**

**}**

```
    public void convertJsonToJava() throws Exception {
        File f = new File("customer.json");

        ObjectMapper mapper = new ObjectMapper();

        // De-Serialization
        Customer customer = mapper.readValue(f, Customer.class);

        System.out.println(customer);
    }
```

```
27      public void convertJavaToJson() throws Exception {
28
29          Customer c = new Customer();
30          c.setId(101);
31          c.setName("Ashok");
32          c.setPhno(9797991);
33          c.setGender("Male");
34
35          File f = new File("customer.json");
36
37          ObjectMapper mapper = new ObjectMapper();
38
39          // serialization
40          mapper.writeValue(f, c);
41
42          System.out.println("Serialization completed.........");
43
44      }
```

## GSON API

**### Git Hub Rep : https://github.com/ashokitschool/GSON-JSON-APP.git**

**<dependency>**

**<groupId>com.google.code.gson</groupId>**

**<artifactId>gson</artifactId>**

**<version>2.3.1</version>**

**</dependency>**

**Gson gson = new Gson ( );**

**gson.toJson(Object obj);**

**gson.fromJson(File file);**

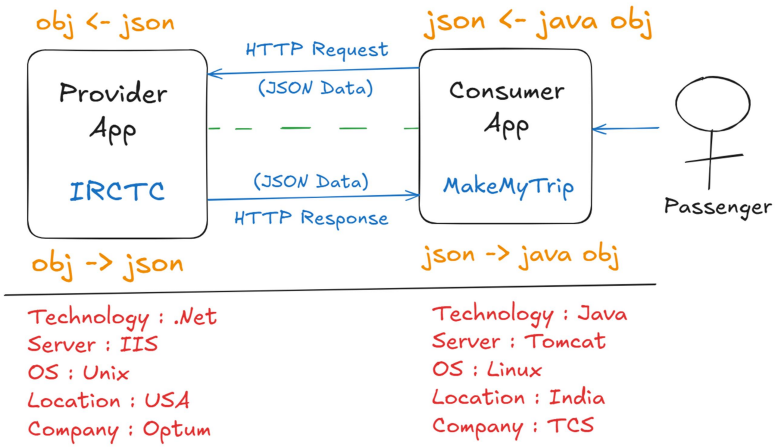
## How to Consumer and Provider will Communicate

**=> End User will communicate with MakeMyTrip application to book Train ticket.**

**=> MakeMyTrip application will take passenger and will send to IRCTC to book train ticket.**

**=> IRCTC will book train ticket and will send ticket information to MakeMyTrip application.**

**=> Make My Trip app will give ticket to end user.**




## HTTP Protocol

**=> HTTP stands for Hyper Text Transfer Protocol.**

**=> HTTP acts as mediator between Client & Server.**

**=> HTTP is stateless protocol.**

**=> HTTP Protocol can't remember the conversation happened between client and server.**

## HTTP Methods

**GET ---------> It is used to get data from server**

**POST ---------> It is used to send data to server**

**PUT ---------> It is used to update the data at server (full update)**

**PATCH ---------> It is used to update partial data**

**DELETE --------> It is used to delete resource at server**

## HTTP Status Codes

**1xx ------ (100 - 199) : Information**

**2xx ------- (200 - 299): Success**

**3xx ------- (300 - 399): Redirection**

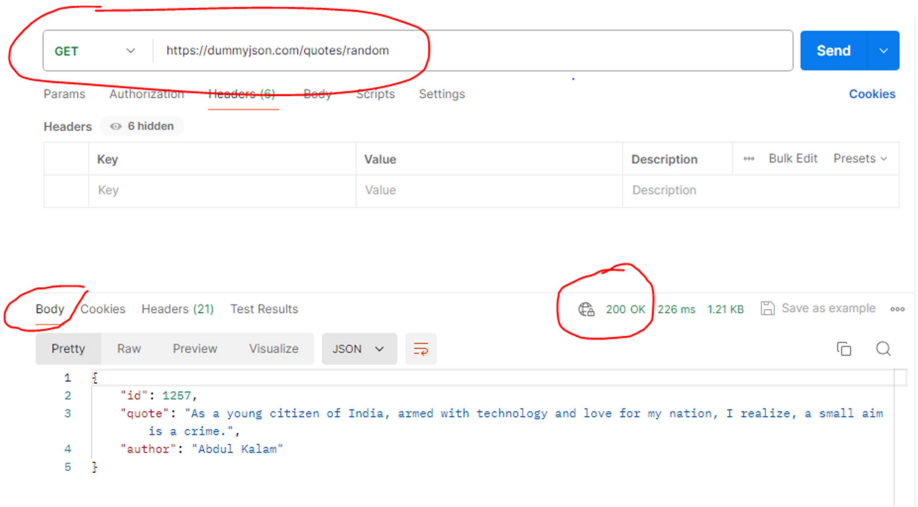
**4xx ------- (400 - 499): Client Error**

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

## What is PostMan

- **Postman is a popular tool used by developers to test APIs (Application Programming Interfaces).**
- **It provides a user-friendly interface for sending HTTP requests, such as GET, POST, PUT, and DELETE, to APIs and receiving responses.**
- **This helps developers to understand how APIs work, debug issues, and ensure that they are functioning correctly.**

**Provider Endpoint : https://dummyjson.com/quotes/random**





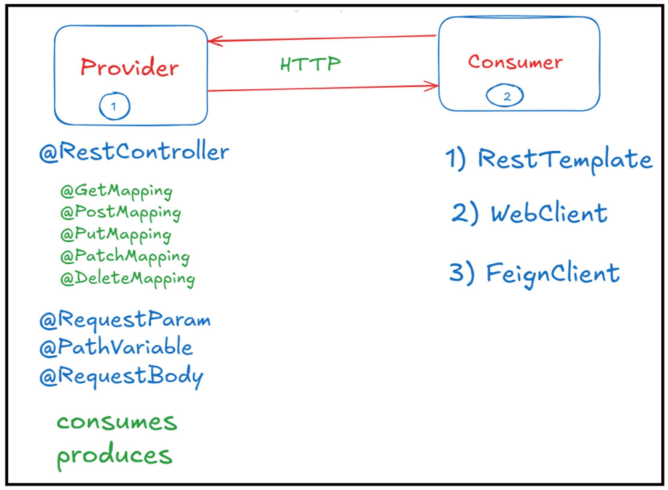

HTTP Request

| | |
|---|---|
| Request Line | HTTP Method + Resource URL |
| Request Headers | Meta Data + Security Tokens |
| Request Body | Business Data (Payload) |

HTTP Response

| | |
|---|---|
| Response line | HTTP Status Code + Status MSG |
| Response headers | Meta Data |
| Response body | Business Data (Payload) |

## Spring Boot Rest API Development Process




- ⇨ **To develop Rest API we will use @RestController annotation**
- ⇨ **@RestController annotation will represent our java class as Distributed Component**

*@RestController = @Controller + @ResponseBody*

- ⇨ **To develop Consumer we have below 3 options**
  - **RestTemplate**
  - **WebClient**
  - **FeignClient**



## Provider (Rest API) Application Development

**Step-1: Create Spring Boot application with below starter**

a) **Web-starter**

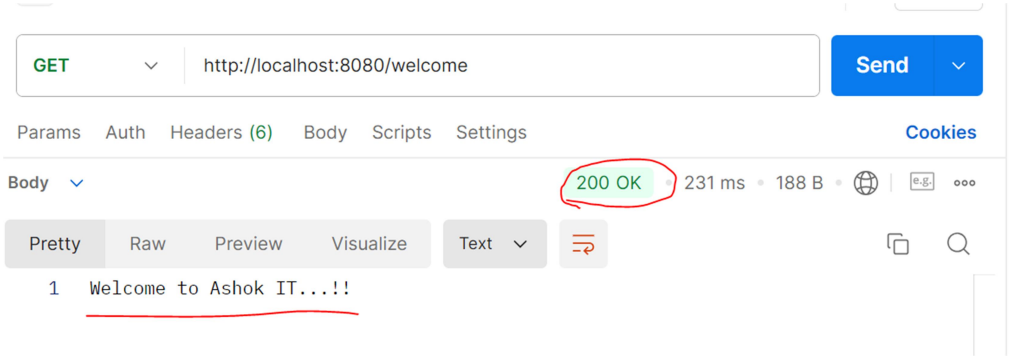
**Step-2: Create Rest Controller class like below**

```
@RestController  no usages
public class MsgRestController {

    @GetMapping(value="/welcome", produces = "text/plain")  no usages
    public String getWelcomeMsg() {
        return "Welcome to Ashok IT... !!";
    }

    @GetMapping(value="/greet", produces = "text/plain")  no usages
    public ResponseEntity<String> getGreetMsg(){
        String msg = "Good Morning";
        return new ResponseEntity<>(msg, HttpStatus.OK);
    }
}
```

**Step-3: Run the application and test it using Postman**




## How to send JSON response to client application

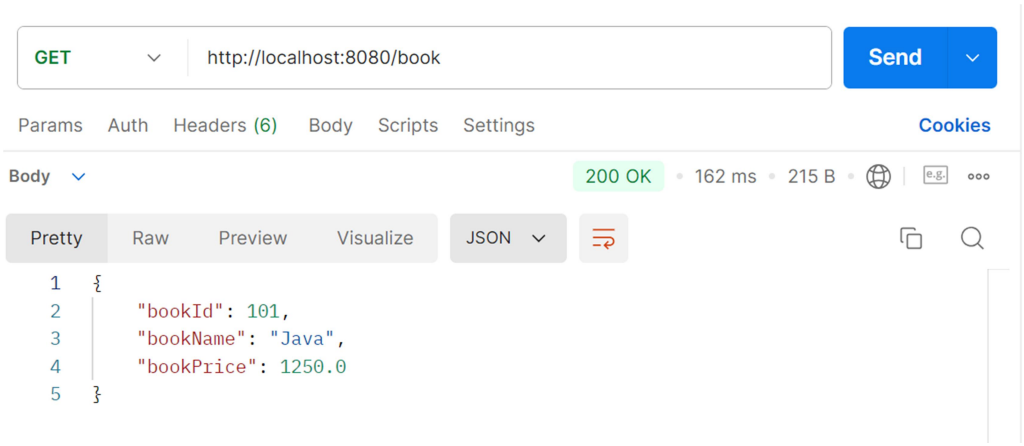
**=> When we return Object (s) from Rest Controller method then springboot internally converts Java Object into JSON using Jackson api and will send JSON response to client.**

**=> In below BookRestController we are returning Object as response**

```
@RestController  no usages
public class BookRestController {

    @GetMapping(value="/book", produces = "application/json")  no usages
    public ResponseEntity<Book> getBook() {
        Book b = new Book( bookId: 101,  bookName: "Java",  bookPrice: 1250.00);
        return new ResponseEntity<>(b, HttpStatus.OK);
    }

    @GetMapping(value="/books", produces = "application/json")  no usages
    public ResponseEntity<List<Book>> getBooks() {
        Book b1 = new Book( bookId: 101,  bookName: "Java",  bookPrice: 1250.00);
        Book b2 = new Book( bookId: 102,  bookName: "Python",  bookPrice: 2250.00);
        Book b3 = new Book( bookId: 103,  bookName: "DevOps",  bookPrice: 3250.00);
        List<Book> booksList = Arrays.asList(b1, b2, b3);
        return new ResponseEntity<>(booksList, HttpStatus.OK);
    }
}
```



## HTTP GET Request

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

**=> GET request will not contain request body.**

**Ex : GET http://www.ashokit.in/courses**

**=> If we want to send data to server using GET request then we need to use**

**1) Query Parameters**

**2) Path Parameters**

**Ex-1(query Param) : https://www.youtube.com/watch?v=W6rCN8Zcikc**

**Ex-2 (Path param) : https://www.youtube.com/shorts/vRzC72VO0qA**

**Ex-4 (Path Param) : https://ashokit.in/courses/java_fullstack_developer**

## What is Query Parameter?

**=> It is used to send data from client to server in URL**

**=> It represents data in key-value format**

**id=101**

**name=Ashok**



**=> It will start with '?' and it will be seperate by '&'**

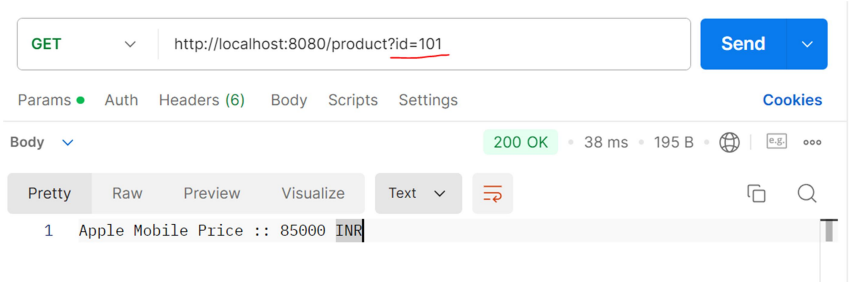
**=> It will present only at end of the URL**

**URL : https://www.youtube.com/watch?v=aNS9G5fyj5Y&t=7190s**

**Note: To read query parameters from URL we will use @RequestParam.**

```java
@RestController  no usages
public class ProductRestController {

    @GetMapping("/product")  no usages
    public ResponseEntity<String> getProductPrice(@RequestParam("id") Integer id) {
        String msg = "";
        if (id == 101) {
            msg = "Apple Mobile Price :: 85000 INR";
        } else if (id == 102) {
            msg = "Samsung Mobile Price :: 65000 INR";
        } else {
            msg = "No Product Found";
        }
        return new ResponseEntity<>(msg, HttpStatus.OK);
    }
}
```




## What is Path Parameter?

**=> It is used to send data from client to server in URL**

**=> Path Parameter will represent data directly (No Key)**

**Ex: https://www.youtube.com/shorts/{vRzC72VO0qA}**

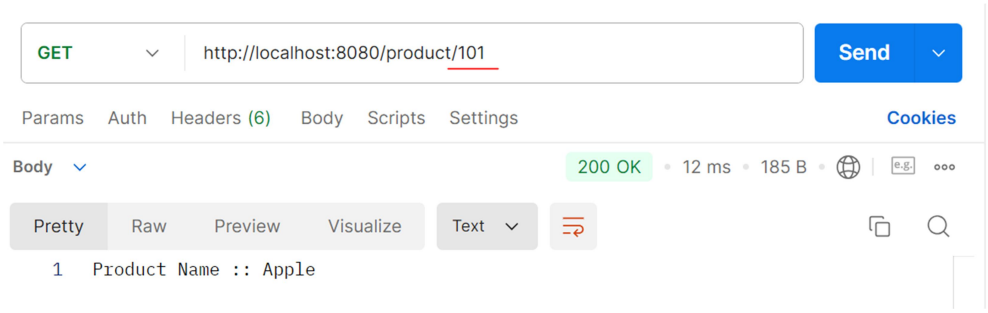
**=> Path Parameter can present anywhere in the URL**

**=> Path Parameter position will be represented in Method URL pattern**

**Note: To read path parameters we will use @PathVariable annotation.**

```
@RestController  no usages
public class ProductRestController {

    @GetMapping("/product/{id}")  no usages
    public ResponseEntity<String> getProductName(@PathVariable("id") Integer id) {
        String msg = "";
        if (id == 101) {
            msg = "Product Name :: Apple";
        } else if (id == 102) {
            msg = "Product Name :: Samsung";
        } else {
            msg = "No Product Found";
        }
        return new ResponseEntity<String>(msg, HttpStatus.OK);
    }
}
```



## HTTP Post Request

**=> HTTP POST method is used to send data from client to server**

**=> We can send payload from client to server in Request Body**

**Ex: json or xml or text or form data or binary data**

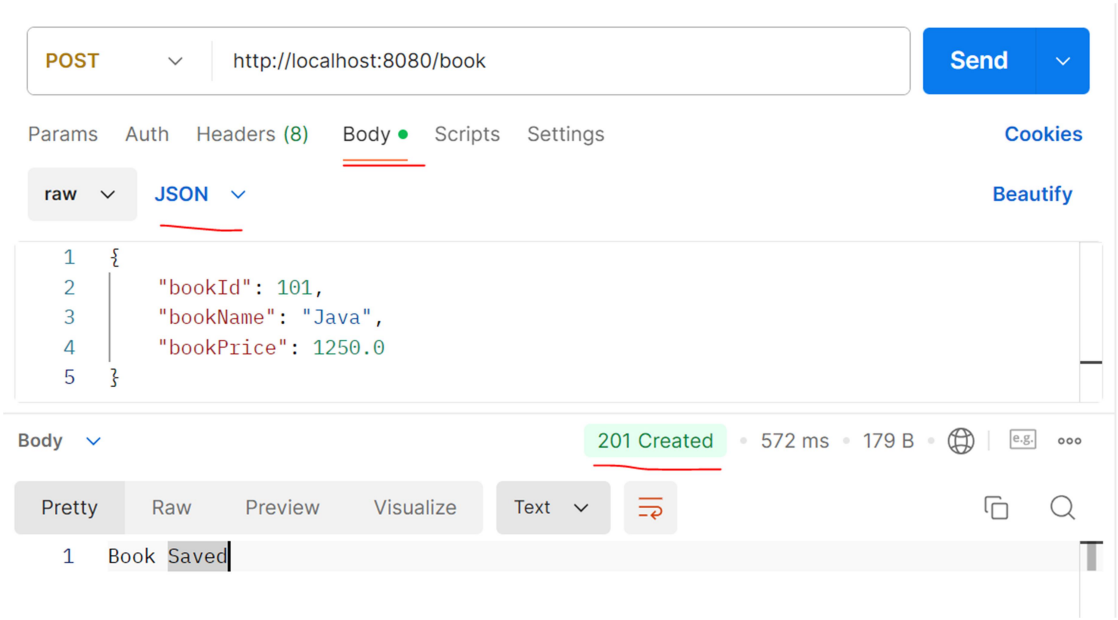
**=> To map our Rest Controller method to POST request we will use @PostMapping annotation**

**Note: To read data from request body we will use @RequestBody annotation**

**produces : Represents rest api method response body data format**

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

```
13
14      @PostMapping(  no usages
15              value = "/book",
16              consumes = "application/json",
17              produces = "text/plain"
18      )
19      public ResponseEntity<String> addBook(@RequestBody Book book) {
20          System.out.println(book);
21          // TODO : save into db
22          String msg = "Book Saved";
23          return new ResponseEntity<>(msg, HttpStatus.CREATED);
24      }
25
```



## HTTP PUT Request

**=> PUT method is used for updating resource/record**

**=> Using HTTP Put Request, we can send data to server in below 3 ways**

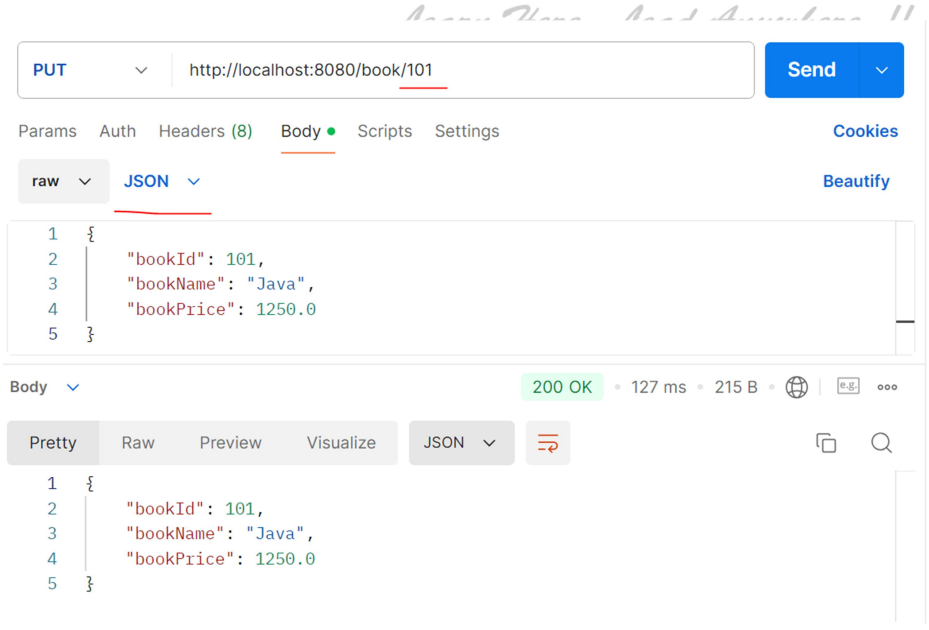
**a) Query Params**

**b) Path params**

**c) Request Body**

**=> To map our Rest Controller method to PUT request we will use @PutMapping annotation.**

```
26      @PutMapping(  no usages
27              value = "/book/{bookId}",
28              consumes = "application/json",
29              produces = "application/json"
30      )
31      public ResponseEntity<Book> updateBook(@PathVariable("bookId") Integer bookId,
32                                             @RequestBody Book book) {
33          System.out.println("Book Id :: " + bookId);
34          System.out.println(book);
35          //TODO: Update Book in DB
36          return new ResponseEntity<>(book, HttpStatus.OK);
37      }
38
```



## HTTP DELETE Request

**=> DELETE method is used for deleting resource/record**

**=> Using HTTP DELETE Request, we can send data to server in below 3 ways**

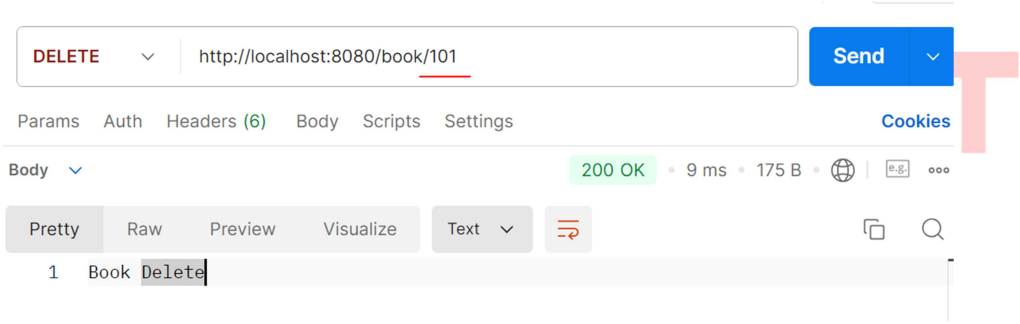
**a) Query Parameters**

**b) Path Parameters**

**c) Request Body**

**=> To map our Rest Controller method to DELETE request we will use @DeleteMapping annotation.**

```
52      @DeleteMapping("/book/{bookId}")  no usages
53      public ResponseEntity<String> deleteBook(@PathVariable("bookId") Integer bookId) {
54          //TODO: delete record from db using id
55          String msg = "Book Delete";
56          return new ResponseEntity<>(msg, HttpStatus.OK);
57      }
```




**Assignment: Develop Spring Boot REST API to perform Crud operations with Database table using Spring Data jpa.**

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

## How to support both XML & JSON data in REST API

To support both xml and json for input and output formats we need to know about below 4 concepts

- Consumes
- Produces
- Accept Header
- Content-Type Header

**Consumes** is used to specify in which format REST API method can accept request payload.

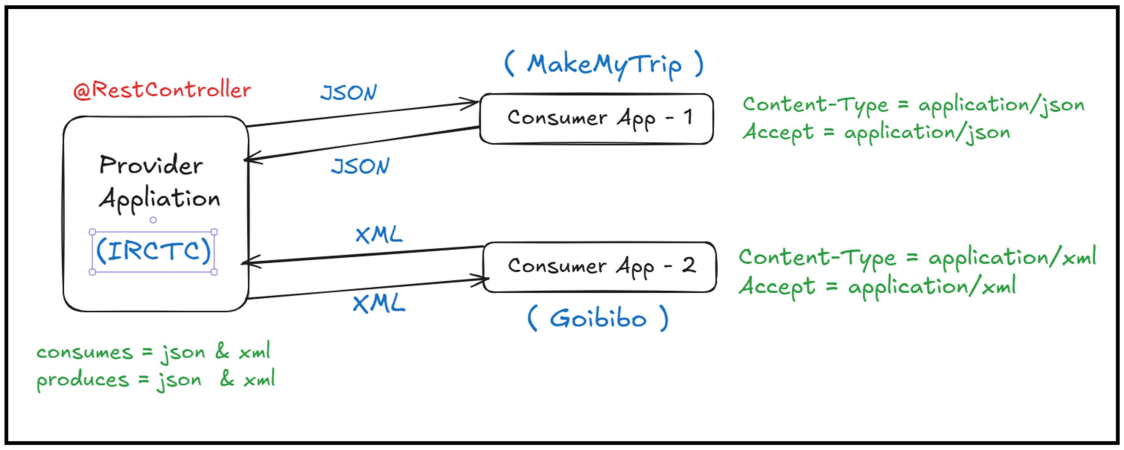
```
consumes = {
        "application/json",
        "application/xml"
},
```

**Produces** is used to specify in which format REST API method can produce response payload.

```
produces = {
        "application/json",
        "application/xml"
}
```

**Content-Type header** is used to specify in which format consumer sending Request payload.

**Accept Header** is used to specify consumer expecting response payload.




Note: If we develop REST API with both xml and json data then it will be more flexible and consumers can decide in which format they want to send and receive data.

## What is XML ?

**-> XML stands for extensible mark-up language**

**-> XML is free and open source**

**-> XML governed by w3c org.**

**-> XML represents data in the form of elements**

**Ex: <name>ashok it</name>**

**-> XML is used to represent the data**

**-> XML is interoperable (language and platform independent)**

```
<Book>
  <bookId>101</bookId>
  <bookName>Java</bookName>
  <bookPrice>1250.0</bookPrice>
</Book>
```

**-> We have 2 types of elements in xml**

**1) Simple element (element which contains data directly)**

**2) Compound element (element which contains child elements)**

## Dealing with xml data in java applications

**=> Up to java 1.8v we have JAX-B api in jdk to deal with xml files in java.**

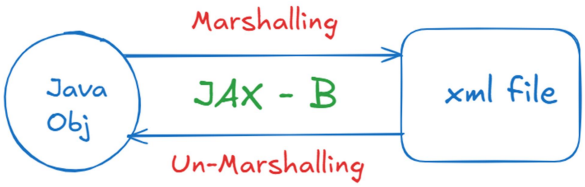
**=> Using JAX-B api we can convert java object to xml and vice versa.**

**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.**

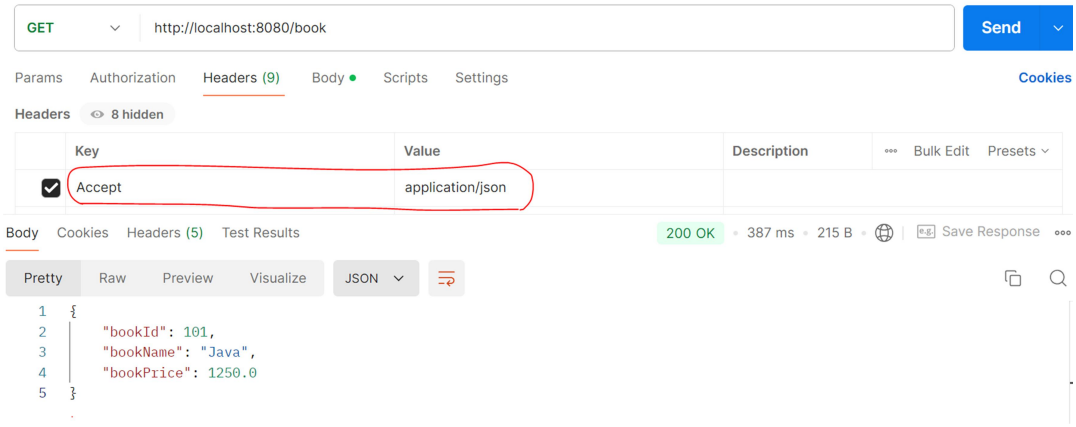
```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

**Below GET request method can produce both json and xml response to consumer.**

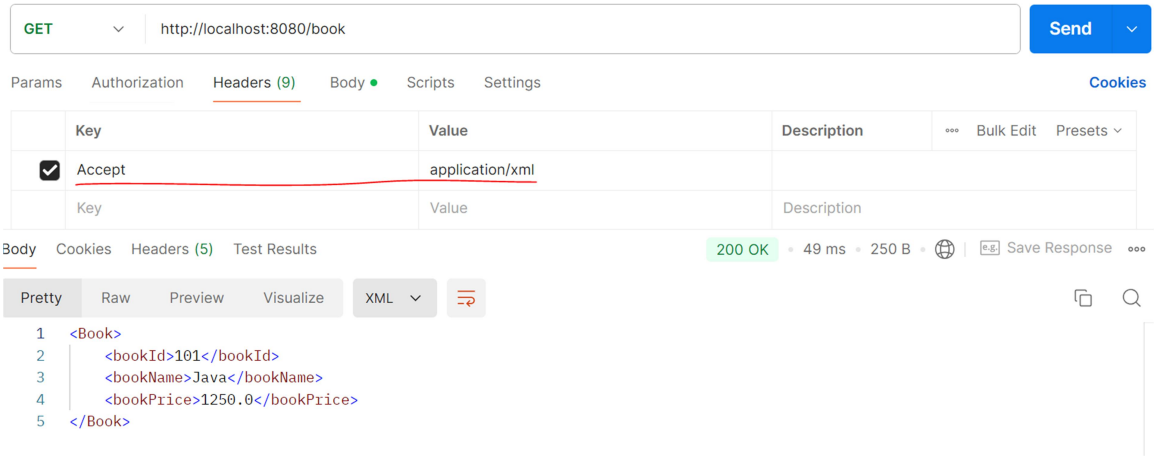
**Consumer can send request with Accept header to receive response in particular format**

```
52    @GetMapping( no usages
53            value = "/book",
54            produces = {
55                    "application/json",
56                    "application/xml"
57            }
58    )
59    public ResponseEntity<Book> getBook() {
60        Book b = new Book( bookId: 101, bookName: "Java", bookPrice: 1250.00);
61        return new ResponseEntity<>(b, HttpStatus.OK);
62    }
63
```

## Sending GET request with Accept = application/json using POSTMAN



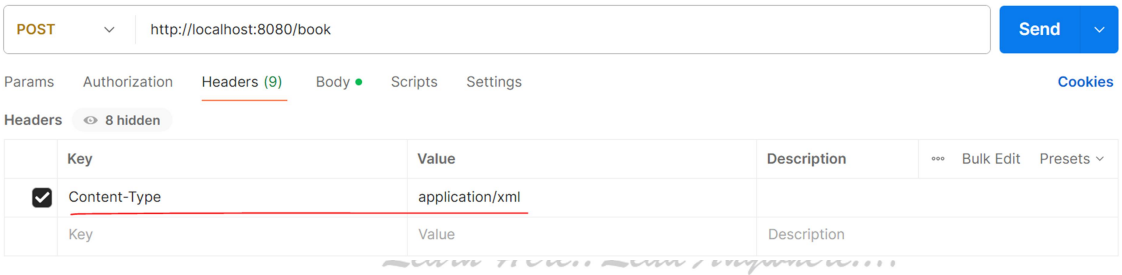

## Sending GET request with Accept = application/xml using POSTMAN



**Below POST request method can consume both xml and json data**

```
@PostMapping(  no usages
        value = "/book",
        consumes = {
                "application/json",
                "application/xml"
        },
        produces = "text/plain"
)
public ResponseEntity<String> addBook(@RequestBody Book book) {
    System.out.println(book);
    // TODO : save into db
    String msg = "Book Saved";
    return new ResponseEntity<>(msg, HttpStatus.CREATED);
}
```

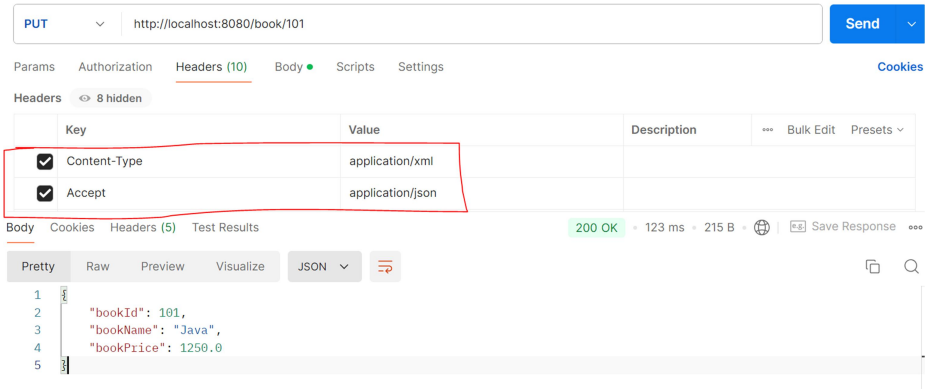
**Sending POST request with XML data in request body using POSTMAN**




**Below PUT request method can consume both xml and json and can produce both xml and json.**

```
@PutMapping(  no usages
        value = "/book/{bookId}",
        consumes =  {"application/json", "application/xml"},
        produces =  {"application/json", "application/xml"}
)
public ResponseEntity<Book> updateBook(@PathVariable("bookId") Integer bookId,
                                       @RequestBody Book book) {
    System.out.println("Book Id :: " + bookId);
    System.out.println(book);
    //TODO: Update Book in DB
    return new ResponseEntity<>(book, HttpStatus.OK);
}
```

**Sending PUT request with both Content-Type and Accept header using POSTMAN**




## What is Swagger ?

=> Swagger is used to generate documentation for REST API

=> Using Swagger UI we can test rest api.

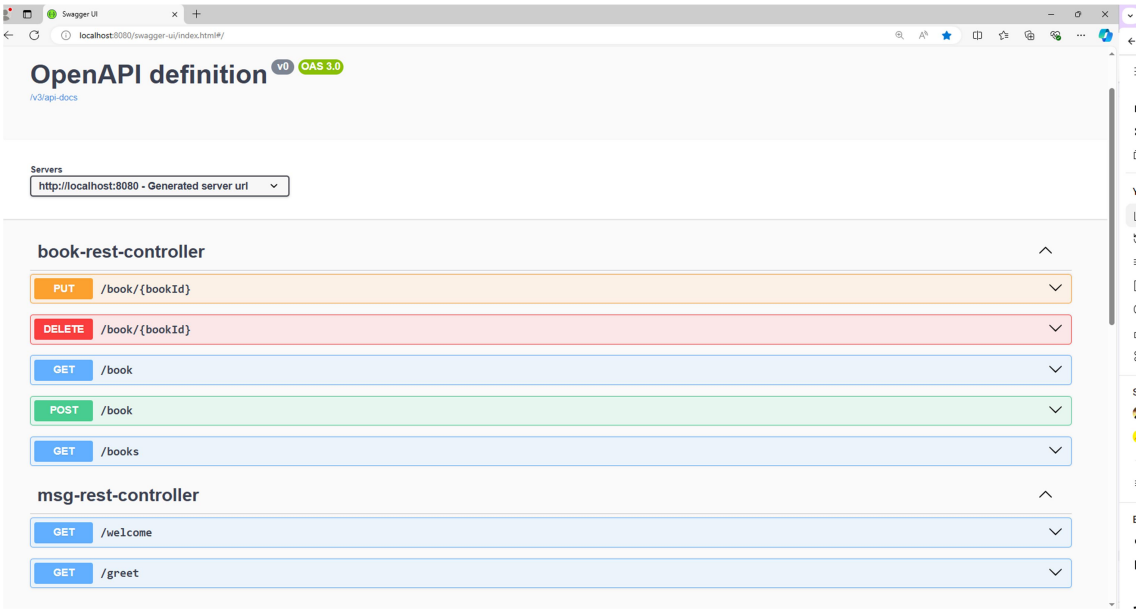
=> Add below dependency in pom.xml file and update maven project.

```
<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 swagger-documentation

URL : http://localhost:8080/swagger-ui.html/



## Consumer Development

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

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

1) Rest Template (sync)

2) Web Client (sync + async)

3) Feign Client

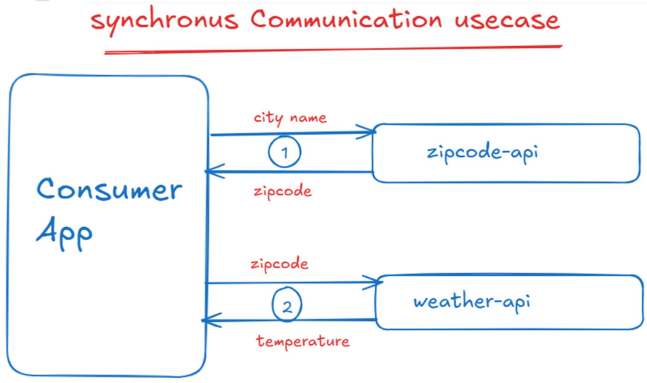
=> "RestTemplate" is a predefined class in 'spring-web-mvc' module. It supports only Synchronus communication.

=> "WebClient" interafce introduced in spring 5.x version and it is part of 'spring-web-flux'. It supports both sync & async communication.

=> FeignClient is part of 'spring-cloud-libraries'. It supports interservice communication.

## What is Synchronus Communication ?

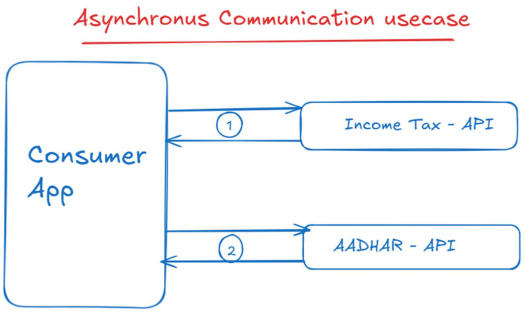
=> After sending request to provider if consumer is waiting for the response then it is called as Synchronus communication.




## What is Asynchronus Communication ?

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

Note : Weather we need to use sync client or async client is depends on project requirement.




## Consumer Application Development

1) Create SpringBoot application with 'web-starter'

2) Create Service class to access provider api using RestTemplate

Provider URL (GET) : https://api.restful-api.dev/objects/

```java
@Service  no usages
public class MsgService {

    private static final String PROVIDER_URL = "https://api.restful-api.dev/objects/";  1 usage

    public void getMsg() {  no usages

        RestTemplate rt = new RestTemplate();
        ResponseEntity<String> forEntity = rt.getForEntity(PROVIDER_URL, String.class);

        HttpStatusCode statusCode = forEntity.getStatusCode();
        String body = forEntity.getBody();

        System.out.println("Status Code : " + statusCode.value());
        System.out.println("Response Body : " + body);

    }
}
```

3) Call the service method in start class for testing.

```java
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        MsgService bean = context.getBean(MsgService.class);

        bean.getMsg();
    }
}
```

## Developing Consumer (MakeMyTrip) application with below functionalities

1) View Tickets
2) Book Ticket

### View Tickets Page design

View Tickets Book Ticket

View All Tickets Here

| Ticket ID | Passenger name | From | To | Journey Date | Ticket Status |
|---|---|---|---|---|---|
| 1 | Ashok | Hyd | Goa | 30-Sep-2024 | CONFIRMED |
| 2 | Raju | Hyd | Pune | 25-Sep-2024 | CONFIRMED |
| 3 | Suresh | Delhi | Mumbai | 18-Sep-2024 | CONFIRMED |

### Book Ticket Page Design

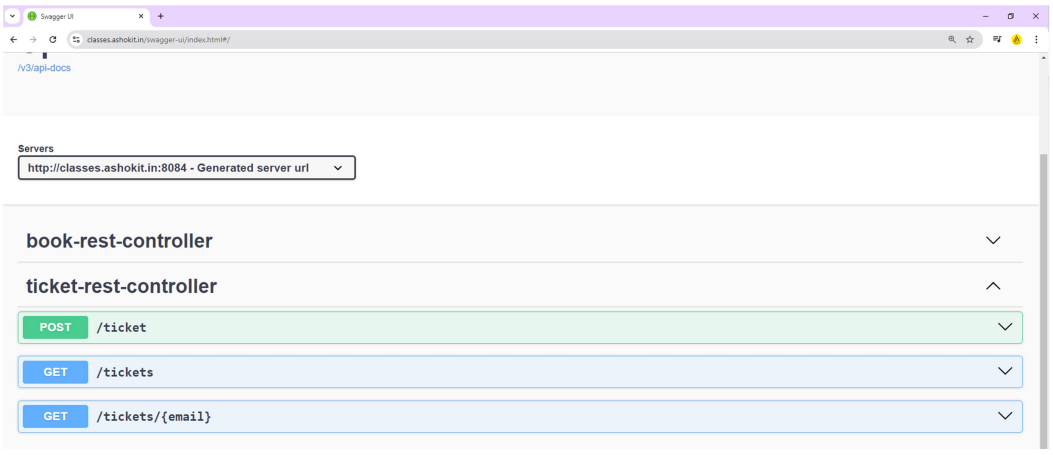
View Tickets Book Ticket

Book Ticket Here

Name :
Email :
From :
To :
DOJ :
Train Num :
Submit

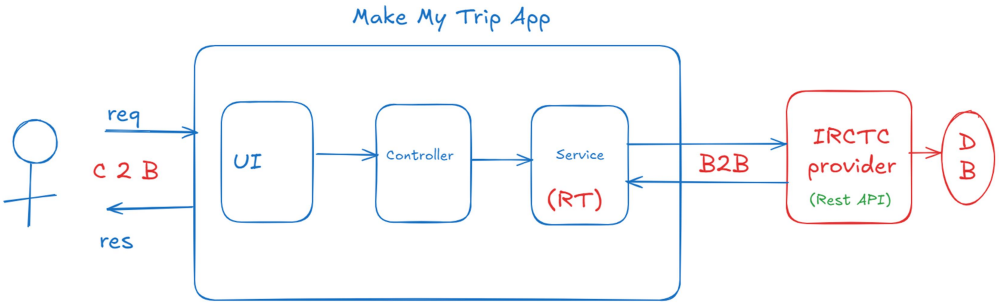


⇨ **Make My Trip application will communicate with IRCTC provider to deal with tickets booking. Below is the provider Swagger Documentation.**



| Swagger Documentation : https://classes.ashokit.in/swagger-ui/index.html |
| --- |
| private static final String BOOK_TICKET_URL = "https://www.classes.ashokit.in/ticket";<br>private static final String GET_TICKETS_URL = "https://www.classes.ashokit.in/tickets"; |

## Consumer Application Architecture and Development Process




1) Understand provider swagger documentation

2) Create Consumer Application with required dependencies

a) web-starter

b) thymeleaf



3) Design Request and Response binding classes to perform B2B communication

Request : Passenger.java

Response : Ticket.java

4) Develop Service component to send request to provider

Method-1 : bookTicket ( )

- input : passenger
- output : ticket

Method-2 : getTickets ( )

- input : NA
- output : tickets

5) Develop Controller with required methods

- index () -> to display all tickets

- loadBookTicketPage () -> display ticketBooking form

- bookTicket() -> handle ticket booking functionailty

6) Create UI pages

- index.html

- bookTicket.html

```
public Ticket bookTicket(Passenger passenger) {  1 usage

    RestTemplate rt = new RestTemplate();

    ResponseEntity<Ticket> ticketResponseEntity =
            rt.postForEntity(BOOK_TICKET_URL, passenger, Ticket.class);

    return  ticketResponseEntity.getBody();
}
```

```
public Ticket[] getAllTickets() {  1 usage

    RestTemplate rt = new RestTemplate();

    ResponseEntity<Ticket[]> forEntity = rt.getForEntity(ALL_TICKETS_URL, Ticket[].class);

    return forEntity.getBody();
}
```

## What is WebClient

=> It is pre-defined interface which is part of 'spring-webflux-starter' (reactive-web)

=> Using WebClient interface we can send HTTP requests to provider applications.

=> WebClient supports both sync & async communications.

```
WebClient webClient = WebClient.create();

Mono<String> mono = webClient.get( )
                                .uri(PROVIDER_API_URL)
                                .retrieve( )
                                .bodyToMono(String.class);

String response = mono.block();

System.out.println(response)
```

## Sending GET request using WebClient (Synchronus)

```
13      public void getQuote1(){  no usages
14          WebClient webClient = WebClient.create();
15
16          Mono<String> mono = webClient.get() RequestHeadersUriSpec<capture of ?>
17                                       .uri(API_URL) capture of ?
18                                       .retrieve() ResponseSpec
19                                       .bodyToMono(String.class);
20
21          //synchronus
22          String response = mono.block();
23          System.out.println(response);
24      }
```

## Sending GET request using WebClient (Synchronus) and map response to Java Object

```
26      public void getQuote2(){  no usages
27          WebClient webClient = WebClient.create();
28
29          Mono<Quote> mono = webClient.get() RequestHeadersUriSpec<capture of ?>
30                                       .uri(API_URL) capture of ?
31                                       .retrieve() ResponseSpec
32                                       .bodyToMono(Quote.class);
33          Quote response = mono.block();
34          System.out.println(response);
35      }
```

## Sending GET request using WebClient (Asynchronus)

```
37      public void getQuote3(){  1 usage
38          WebClient webClient = WebClient.create();
39          System.out.println("======== Request sending - started ========");
40          webClient.get() RequestHeadersUriSpec<capture of ?>
41                  .uri(API_URL) capture of ?
42                  .retrieve() ResponseSpec
43                  .bodyToMono(Quote.class) Mono<Quote>
44                  .subscribe(response → {
45                      // handle response
46                      handleResponse(response);
47                  });
48          System.out.println("========== Request sending - completed ======= ");
49      }
50      private void handleResponse(Quote response){  1 usage
51          System.out.println(response);
52      }
```

## Sending POST request using WebClient

```
public Mono<Ticket> bookTicket(Passenger passenger) {

    WebClient webClient = WebClient.create();

    Mono<Ticket> bodyToMono = webClient.post()
                                        .uri(apiUrl)
                                        .body(BodyInserters.fromValue(passenger))
                                        .retrieve()
                                        .bodyToMono(Ticket.class);
    return bodyToMono;

}
```