REST WEBSERVICES

Document for Java REST Web Services: A Comprehensive Guide

1. Introduction to REST Web Services

What are REST Web Services?

- REST (Representational State Transfer) is an architectural style for designing networked applications.
- RESTful services use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources.
- They are stateless and communicate primarily via JSON or XML.

Why Use REST Web Services?

- Platform-independent communication (works across languages/devices).
- Lightweight compared to SOAP.
- Easy to scale and cache.
- Standard HTTP methods (GET, POST, PUT, DELETE) are used, making APIs intuitive.
- Wide adoption and excellent tooling support.

2. Core Concepts of REST Web Services

Resources and URLs

- Everything is a resource (e.g., persons, states, Login).
- Resources are identified by URLs (Uniform Resource Locators).
- Example: /persons represents persons resources.
- `/users` — collection of users/persons
- `/users/{id}` — specific user/person by ID
- Each request from client to server must contain all necessary info.
- Server doesn't store client sessions.

Resource	HTTP Method	Purpose
/users	GET	List all users
/users	POST	Create a new user
/users/{id}	GET	Get user details by ID
/users/{id}	PUT	Update user by ID
/users/{id}	DELETE	Delete user by ID

Example:

Endpoint	Method	Description
/users	GET	Retrieve all users
/users/{userId}	GET	Retrieve single user by ID

3. Input and Output Formats

Common Data Formats

- JSON (JavaScript Object Notation) --- most used format
- XML (Extensible Markup Language).
- Plain text or HTML (less common).

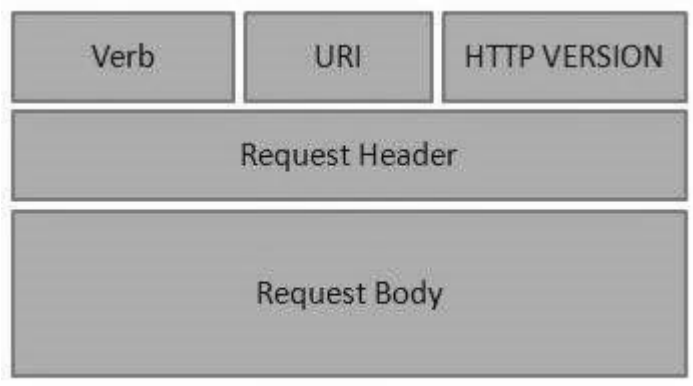
4. Maven Project Structure and Purpose of Each Layer

```
my-rest-api/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com.example.restapi/
│   │   │       ├── controller/      <-- Exposes REST endpoints
│   │   │       ├── model/           <-- Entity classes (data)
│   │   │       ├── service/         <-- Business logic
│   │   │       ├── repository/      <-- DB operations (Spring Data JPA)
│   │   │       └── RestApiApplication.java <-- Main class
│   │   └── resources/
│   │       ├── application.properties <-- App config (DB, port, etc.)
├── pom.xml                          <-- Maven configuration
```

5. REST Messaging:

- 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.

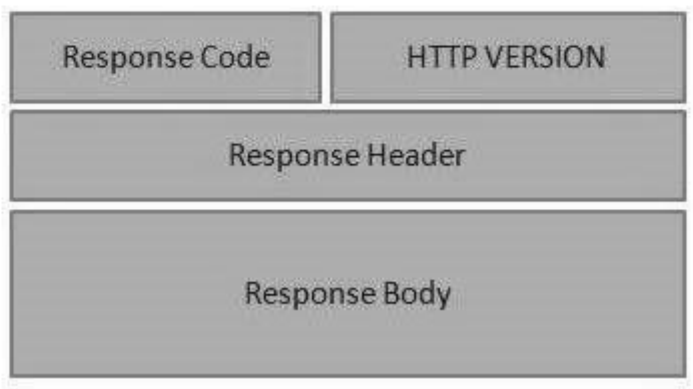
HTTP Request



HTTP Request

- **Verb:** Indicated the HTTP methods such as GET, POST, PUT, DELETE etc.
- **Uri:** Uniform resource identifier to identify the resource on the server.
- **HTTP Version:** Indicates the version of the http
- **Request Header:** Contain metadata for the HTTP Request message as Key-Value pairs.
- **Request Body:** Message content or Request representation.

HTTP Response



HTTP Response

- Status/Response code: Indicates the server status of the request response. Like 400,200 ,404 etc.

- Response Headers: Contains the metadata for the HTTP Response as Key-Value pairs.
- Response Body: Response message content or Resource representation.

6. REST Methods:

❖ **GET**

- **Purpose:** Retrieve data from the server.
- **Request body:** Usually no request body
- **Response:** Returns data in JSON, XML, HTML.

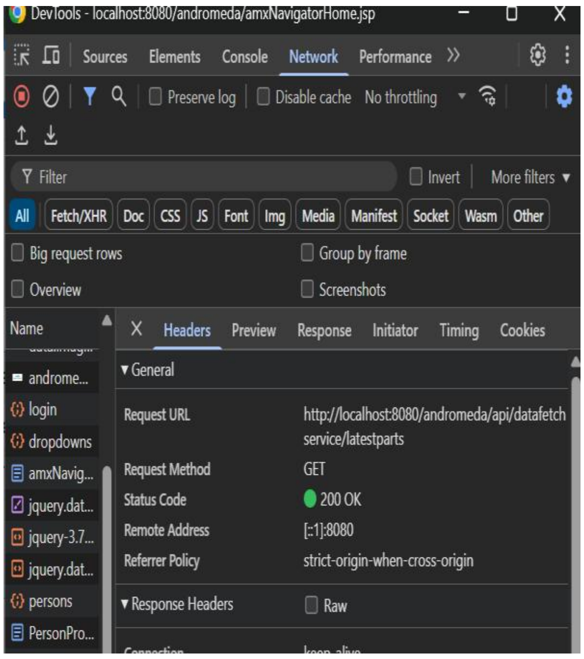
Example:

@GET ---- annotation

Request Method: GET

Accept: Application/Json

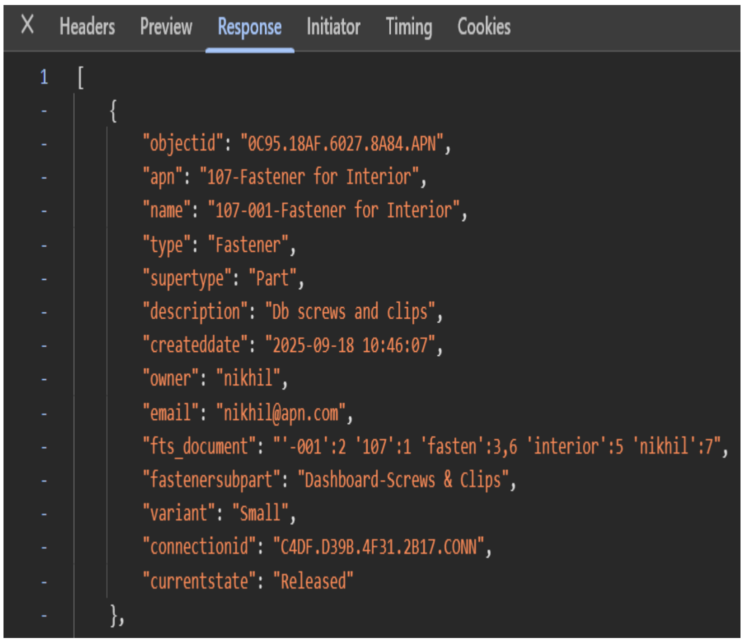
Response: Json Format



▾ General

Request URL http://localhost:8080/andromeda/api/datafetchservice/latestparts
Request Method GET
Status Code ● 200 OK
Remote Address [::1]:8080
Referrer Policy strict-origin-when-cross-origin

▾ Response Headers ☐ Raw



```
[
    {
        "objectid": "0C95.18AF.6027.8A84.APN",
        "apn": "107-Fastener for Interior",
        "name": "107-001-Fastener for Interior",
        "type": "Fastener",
        "supertype": "Part",
        "description": "Db screws and clips",
        "createddate": "2025-09-18 10:46:07",
        "owner": "nikhil",
        "email": "nikhil@apn.com",
        "fts_document": "'-001':2 '107':1 'fasten':3,6 'interior':5 'nikhil':7",
        "fastenersubpart": "Dashboard-Screws & Clips",
        "variant": "Small",
        "connectionid": "C4DF.D39B.4F31.2B17.CONN",
        "currentstate": "Released"
    },
```

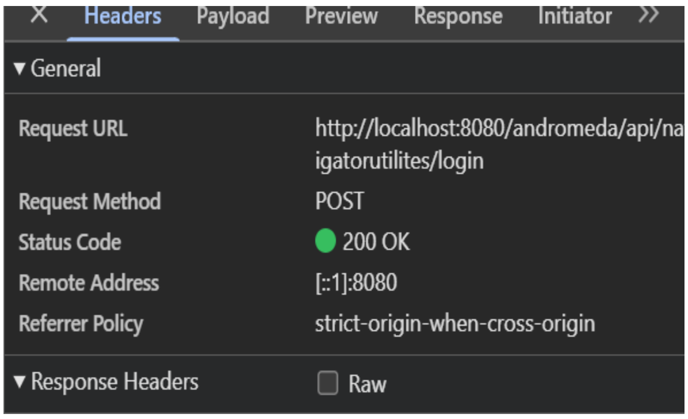
❖ **POST**

- **Purpose:** Create a new resource on the server
- **Request body:** Contains the data to create the resource
- **Response:** Returns the created message or often status code with a 201 created status and a locator header pointing to new resource.

Example:

@POST

Content-Type: Application/Json

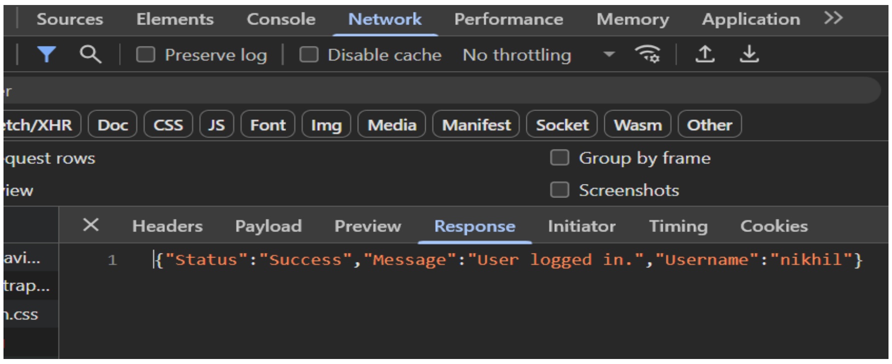


▾ General

Request URL http://localhost:8080/andromeda/api/naigatorutilites/login
Request Method POST
Status Code ● 200 OK
Remote Address [::1]:8080
Referrer Policy strict-origin-when-cross-origin

▾ Response Headers ☐ Raw

Response: Json Format
{"Status":"Success","Message":"User logged in.","Username":"nikhil"}

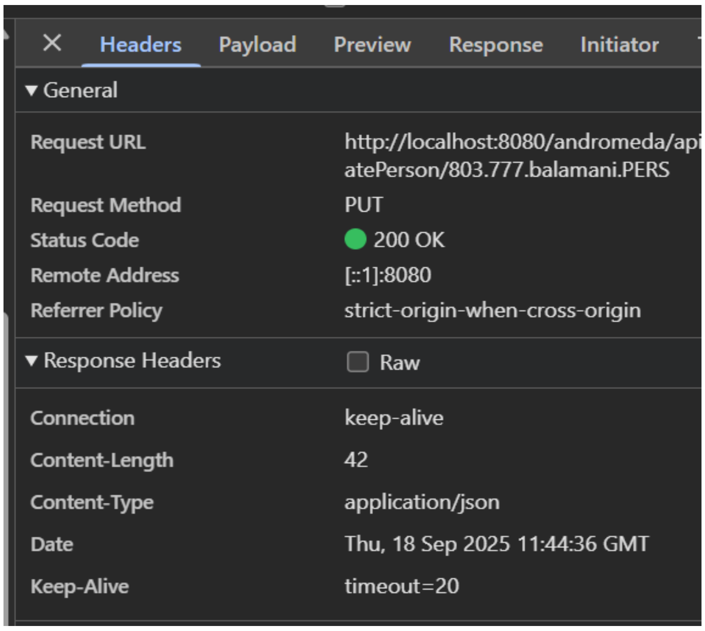

Sources Elements Console Network Performance Memory Application
Preserve log Disable cache No throttling
tch/XHR Doc CSS JS Font Img Media Manifest Socket Wasm Other
quest rows
Group by frame
iew
Screenshots
Headers Payload Preview Response Initiator Timing Cookies

```
{"Status":"Success","Message":"User logged in.","Username":"nikhil"}
```

❖ **PUT**

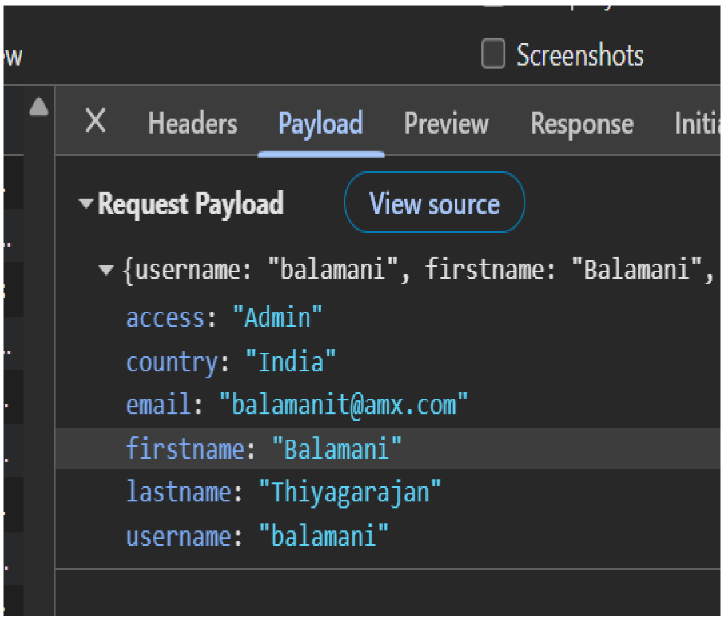
- **Purpose:** Update an existing resource or create it if it doesn't exist.
- **Request body:** Contains the full updated resource representation.
- **Response:** Usually returns the updated resource or a status message.

Example:
@PUT
Content-Type: Application/Json



```
Request URL        http://localhost:8080/andromeda/api
                   atePerson/803.777.balamani.PERS
Request Method     PUT
Status Code        200 OK
Remote Address     [::1]:8080
Referrer Policy    strict-origin-when-cross-origin

Connection         keep-alive
Content-Length     42
Content-Type       application/json
Date               Thu, 18 Sep 2025 11:44:36 GMT
Keep-Alive         timeout=20
```

Request body:



```
{username: "balamani", firstname: "Balamani",
  access: "Admin"
  country: "India"
  email: "balamanit@amx.com"
  firstname: "Balamani"
  lastname: "Thiyagarajan"
  username: "balamani"
```

Response message:
{message: "Person updated successfully"}
message: "Person updated successfully"

❖ **PATCH**

- **Purpose:** Partially update a resource (send only the changes)
- **Request body:** Contains partial data to update the resource.
- **Response:** Returns the updated resource or a status message.

Example:
@PATCH
Content-Type: Application/Json
Request body:
{
"email": "john.new@example.com"}

Response:
HTTP/1.1 200 OK

```
{
  "id": 123,
  "name": "John Smith",
  "email": "john.new@example.com"
}
```

- ❖ **DELETE**
- **Purpose:** Delete a resource
- **Request body:** Usually none. If need to delete a specific field that time pass the request based on what user what to delete.
- **Response:** Deletes the record/resource and gives a status code or some message.

Example:
@DELETE
Content-Type: Application/Json
Request body: For specific record using nid
nid   NID00000002
Response: It deletes the specific record based on the nid
{“Status”: “Success”, “Message”: “Record Deleted”}

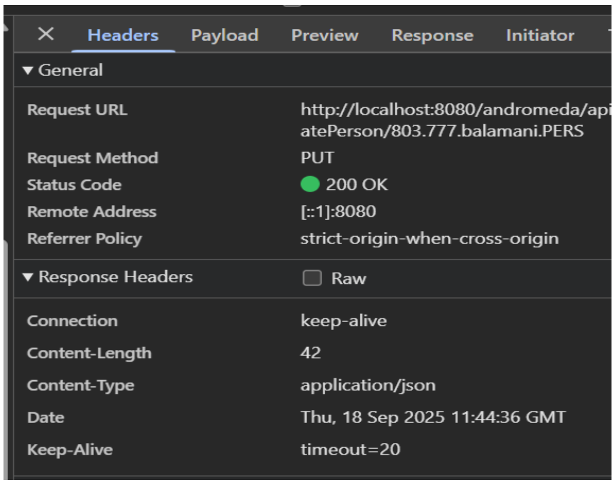
7. STATUS CODES:
HTTP status codes are three-digit codes returned by the server to indicate the result of a client’s request. They are grouped into five classes:

7.1. 1xx – Informational Status code:
These are rarely used in REST APIs and mostly handled by HTTP protocol itself.

- **100 Continue:** The initial part of a request has been received and the client should continue.
- **101 Switching Protocols:** Server is switching protocols as requested by the client.

7.2. 2xx – Success Status code:
Indicates that the client’s request was successfully received, understood, and accepted. Common 2xx status codes in Rest.

- ❖ **200 – OK:** Request succeeded, and response body contains the requested data (e.g., GET).
- ❖ **201 – Created:** A new resource was successfully created (e.g., POST). Location header should point to new resource URI**.**
- ❖ **202 – Accepted:** Request accepted but not yet processed (e.g., async operations).
- ❖ **204 – No Content**: Request succeeded but no content to return (e.g., successful DELETE or PUT with no response body). Example of Success status code.



▾ General

Request URL	http://localhost:8080/andromeda/api atePerson/803.777.balamani.PERS
Request Method	PUT
Status Code	● 200 OK
Remote Address	[::1]:8080
Referrer Policy	strict-origin-when-cross-origin

▾ Response Headers ☐ Raw

Connection	keep-alive
Content-Length	42
Content-Type	application/json
Date	Thu, 18 Sep 2025 11:44:36 GMT
Keep-Alive	timeout=20

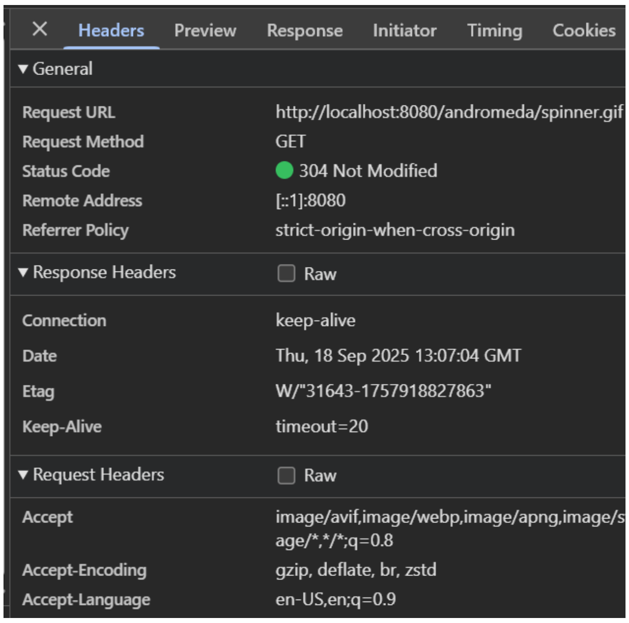
7.3. 3xx – Redirection Status code:

These codes indicate the client needs to take additional action to complete the request.

- ❖ **301 –Moved Permanently:** The requested resource has been permanently moved to a new URI.
- ❖ **302 –Found (Temporary Redirect):** The resource temporarily resides under a different URI.
- ❖ **304 –Not Modified:** Used with caching to indicate resource hasn't changed.

In REST APIs, these are rarely used directly by clients.

Example of Redirection Status code:



▾ General

```
Request URL       http://localhost:8080/andromeda/spinner.gif
Request Method    GET
Status Code       ● 304 Not Modified
Remote Address    [::1]:8080
Referrer Policy   strict-origin-when-cross-origin
```

▾ Response Headers ☐ Raw

```
Connection        keep-alive
Date              Thu, 18 Sep 2025 13:07:04 GMT
Etag              W/"31643-1757918827863"
Keep-Alive        timeout=20
```

▾ Request Headers ☐ Raw

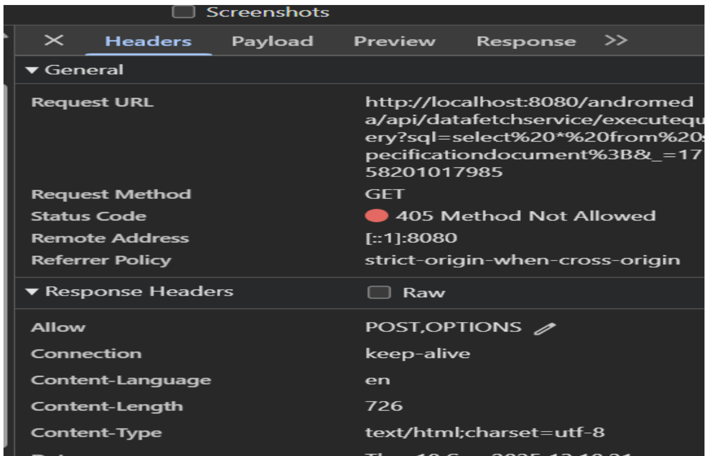
```
Accept            image/avif,image/webp,image/apng,image/s
                  age/*,*/*;q=0.8
Accept-Encoding   gzip, deflate, br, zstd
Accept-Language   en-US,en;q=0.9
```

7.4. 4xx – Client Errors Status code:

Indicates issues with the client's request. These are the most important to communicate errors clearly.

- ❖ **400–Bad Request:** The request is malformed or invalid (e.g., missing parameters, invalid JSON).
- ❖ **401–Unauthorized:** Authentication is required or has failed (e.g., missing or invalid token).
- ❖ **403–Forbidden:** The client is authenticated but does not have permission to access the resource.
- ❖ **404–Not Found:** The requested resource does not exist.
- ❖ **405–Method Not Allowed:** The HTTP method used is not supported by this resource (e.g., POST on a GET-only endpoint).
- ❖ **406–Not Acceptable:** The requested format is not supported (e.g., client requests XML but only JSON is supported).
- ❖ **409–Conflict:** The request could not be completed due to a conflict (e.g., duplicate resource).
- ❖ **415–Not Supported Media Type:** The request payload format is unsupported (e.g., sending XML when only JSON accepted).
- ❖ 429–**To Many Requests**: The client has sent too many requests in a given amount of time (rate limiting).

Example of Client Error Status code:



▾ General

```
Request URL       http://localhost:8080/andromed
                  a/api/datafetchservice/executequ
                  ery?sql=select%20*%20from%20s
                  pecificationdocument%3B&_=17
                  58201017985
Request Method    GET
Status Code       ● 405 Method Not Allowed
Remote Address    [::1]:8080
Referrer Policy   strict-origin-when-cross-origin
```

▾ Response Headers ☐ Raw

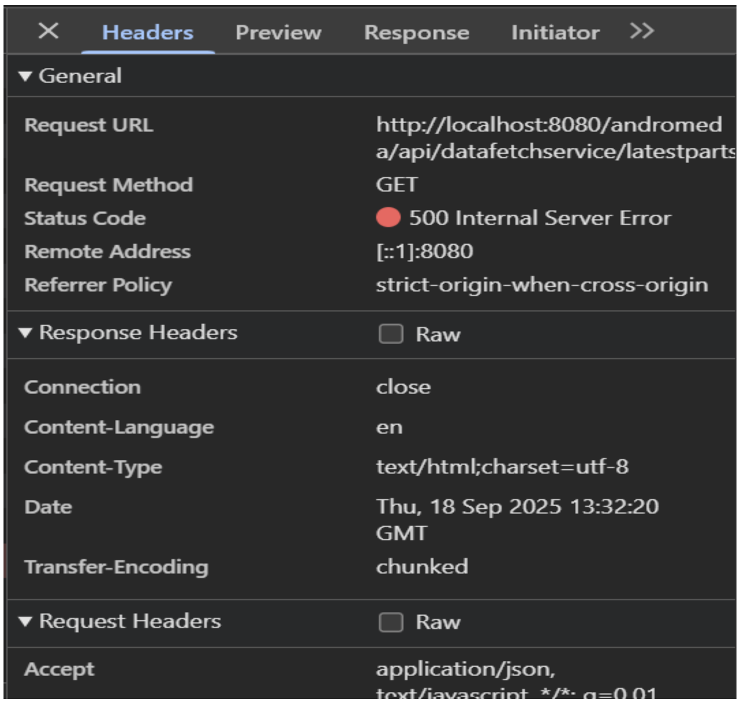
```
Allow             POST,OPTIONS
Connection        keep-alive
Content-Language  en
Content-Length    726
Content-Type      text/html;charset=utf-8
```

7.5. 5xx – Server Errors Status codes:

These indicate that the server failed to fulfil a valid request due to an error on its side.

- ❖ **500–Internal Server Issue:** Generic server error, something unexpected happened.
- ❖ **501–Not Implemented**: The server does not support the functionality required to fulfil the request.
- ❖ **502–Bad Gateway:** Server received an invalid response from an upstream server.
- ❖ **503–Service Unavailable:** Server is currently unable to handle the request (e.g., maintenance, overload).
- ❖ **504–Gateway Timeout:** Server did not receive a timely response from upstream server.

Example of Server Error status code:



▾ General

Request URL	http://localhost:8080/andromeda/api/datafetchservice/latestparts
Request Method	GET
Status Code	🔴 500 Internal Server Error
Remote Address	[::1]:8080
Referrer Policy	strict-origin-when-cross-origin

▾ Response Headers ☐ Raw

Connection	close
Content-Language	en
Content-Type	text/html;charset=utf-8
Date	Thu, 18 Sep 2025 13:32:20 GMT
Transfer-Encoding	chunked

▾ Request Headers ☐ Raw

Accept	application/json, text/javascript, */*; q=0.01

8. JAX-RS REST Web Services Annotations

i. @PATH:

Purpose: Defines the relative URI path for a REST resource class or a specific method.

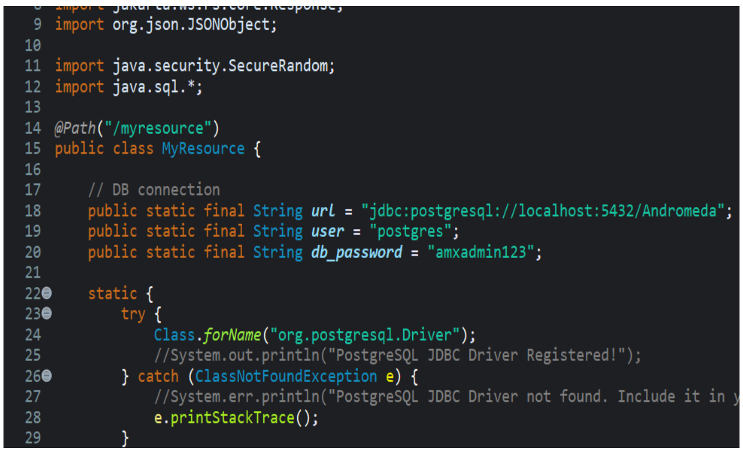
Where to use? On a class or a method

Details:

- When applied on a class, it defines the base URI for all the resource methods in that class.
- When applied on a method, it appends to the class-level path for more specific URIs.

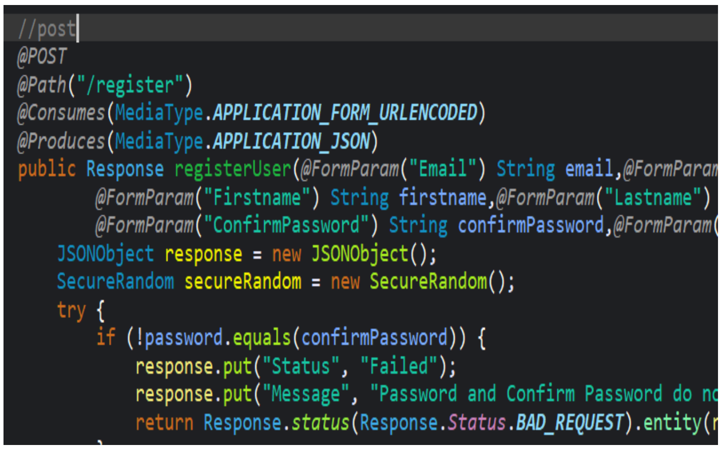
Example:

1.On a class



```java
import org.json.JSONObject;

import java.security.SecureRandom;
import java.sql.*;

@Path("/myresource")
public class MyResource {

    // DB connection
    public static final String url = "jdbc:postgresql://localhost:5432/Andromeda";
    public static final String user = "postgres";
    public static final String db_password = "amxadmin123";

    static {
        try {
            Class.forName("org.postgresql.Driver");
            //System.out.println("PostgreSQL JDBC Driver Registered!");
        } catch (ClassNotFoundException e) {
            //System.err.println("PostgreSQL JDBC Driver not found. Include it in y
            e.printStackTrace();
        }
```

2. On a method



```java
//post
@POST
@Path("/register")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response registerUser(@FormParam("Email") String email,@FormParam
        @FormParam("Firstname") String firstname,@FormParam("Lastname")
        @FormParam("ConfirmPassword") String confirmPassword,@FormParam(
    JSONObject response = new JSONObject();
    SecureRandom secureRandom = new SecureRandom();
    try {
        if (!password.equals(confirmPassword)) {
            response.put("Status", "Failed");
            response.put("Message", "Password and Confirm Password do no
            return Response.status(Response.Status.BAD_REQUEST).entity(r
```

Here, @Path("/myresource") defines the base URI of a class, and the method handles @Path("/register")

ii. **@GET**

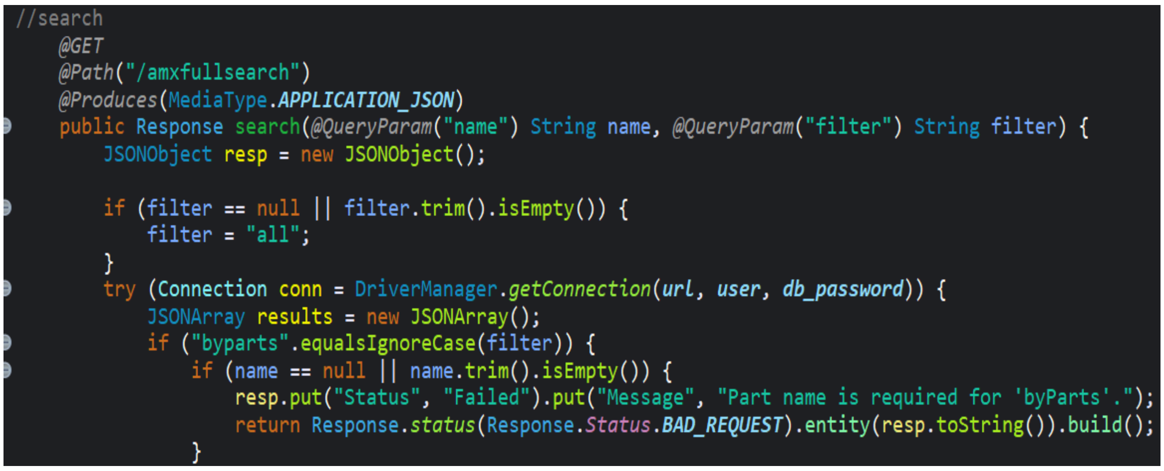
Purpose: Maps an HTTP **GET** request to the annotated method.

Where to use: On a method.

Details:

- Used for reading/fetching resources.
- Should not modify server state.

Example:



```java
//search
    @GET
    @Path("/amxfullsearch")
    @Produces(MediaType.APPLICATION_JSON)
    public Response search(@QueryParam("name") String name, @QueryParam("filter") String filter) {
        JSONObject resp = new JSONObject();

        if (filter == null || filter.trim().isEmpty()) {
            filter = "all";
        }
        try (Connection conn = DriverManager.getConnection(url, user, db_password)) {
            JSONArray results = new JSONArray();
            if ("byparts".equalsIgnoreCase(filter)) {
                if (name == null || name.trim().isEmpty()) {
                    resp.put("Status", "Failed").put("Message", "Part name is required for 'byParts'.");
                    return Response.status(Response.Status.BAD_REQUEST).entity(resp.toString()).build();
                }
```

iii. **@POST**

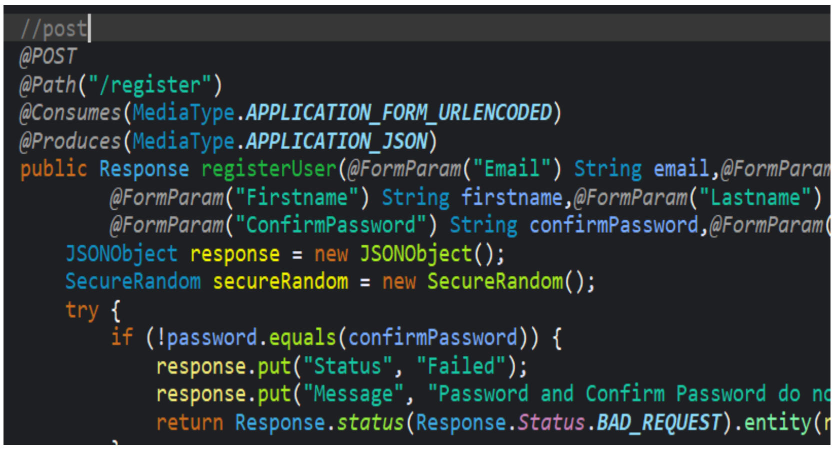
Purpose: Maps an HTTP **POST** request to the annotated method.

Where to use: On a method.

Details:

- Usually used to **create** a new resource or trigger some processing.
- Accepts data in the request body.

Example:



```java
//post
@POST
@Path("/register")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response registerUser(@FormParam("Email") String email,@FormParam
        @FormParam("Firstname") String firstname,@FormParam("Lastname")
        @FormParam("ConfirmPassword") String confirmPassword,@FormParam(
    JSONObject response = new JSONObject();
    SecureRandom secureRandom = new SecureRandom();
    try {
        if (!password.equals(confirmPassword)) {
            response.put("Status", "Failed");
            response.put("Message", "Password and Confirm Password do n
            return Response.status(Response.Status.BAD_REQUEST).entity(r
```

iv. **@PUT**

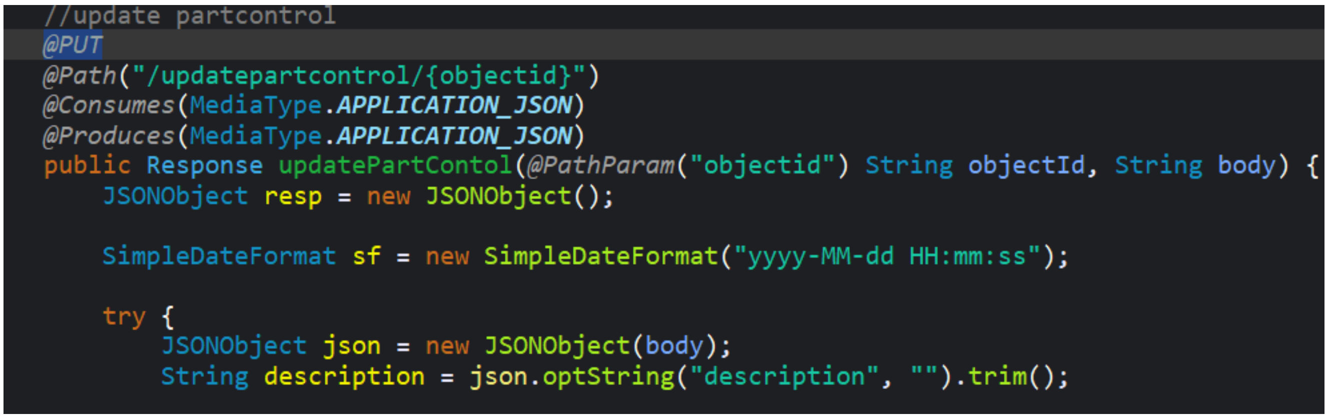
Purpose: Maps an HTTP PUT request to the annotated method.

Where to use: On a method.

Details:

- Commonly used to update a resource or create it if it doesn't exist.
- Idempotent operation.

Example:



```java
//update partcontrol
@PUT
@Path("/updatepartcontrol/{objectid}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response updatePartContol(@PathParam("objectid") String objectId, String body) {
    JSONObject resp = new JSONObject();

    SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    try {
        JSONObject json = new JSONObject(body);
        String description = json.optString("description", "").trim();
```

v. **@DELETE**

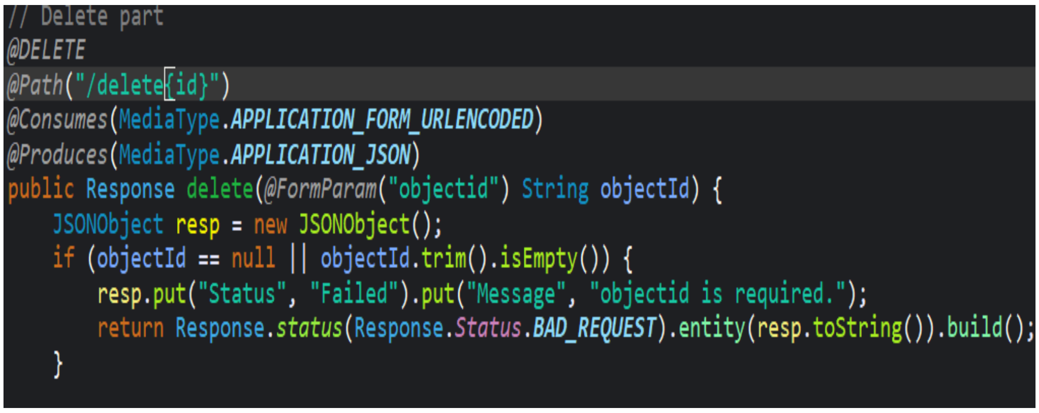
Purpose: Maps an HTTP **DELETE** request to the annotated method.

Where to use: On a method.

Details:

- Used to delete a resource.

Example:



```java
// Delete part
@DELETE
@Path("/delete{id}")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response delete(@FormParam("objectid") String objectId) {
    JSONObject resp = new JSONObject();
    if (objectId == null || objectId.trim().isEmpty()) {
        resp.put("Status", "Failed").put("Message", "objectid is required.");
        return Response.status(Response.Status.BAD_REQUEST).entity(resp.toString()).build();
    }
```

vi. **@Produces**

Purpose: Specifies the media types (MIME types) the method or resource can **produce** in the response.

Where to use: On a method or class.

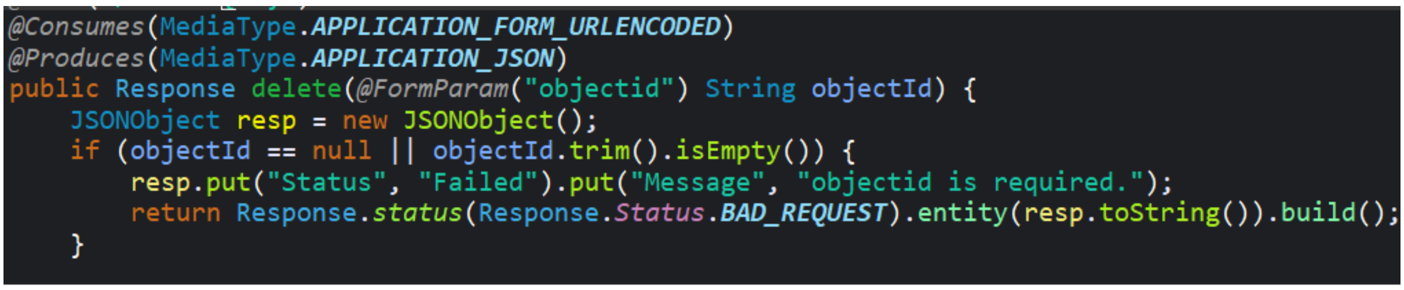
Details:

- Defines the **Content-Type** returned by the resource.
- Can specify multiple types (e.g., JSON, XML, plain text).

Common media types:

- MediaType.APPLICATION_JSON (application/Json)
- MediaType.APPLICATION_XML (application/xml)
- MediaType.TEXT_PLAIN (text/plain)

Example:



```java
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response delete(@FormParam("objectid") String objectId) {
    JSONObject resp = new JSONObject();
    if (objectId == null || objectId.trim().isEmpty()) {
        resp.put("Status", "Failed").put("Message", "objectid is required.");
        return Response.status(Response.Status.BAD_REQUEST).entity(resp.toString()).build();
    }
```

vii. **@Consumes**

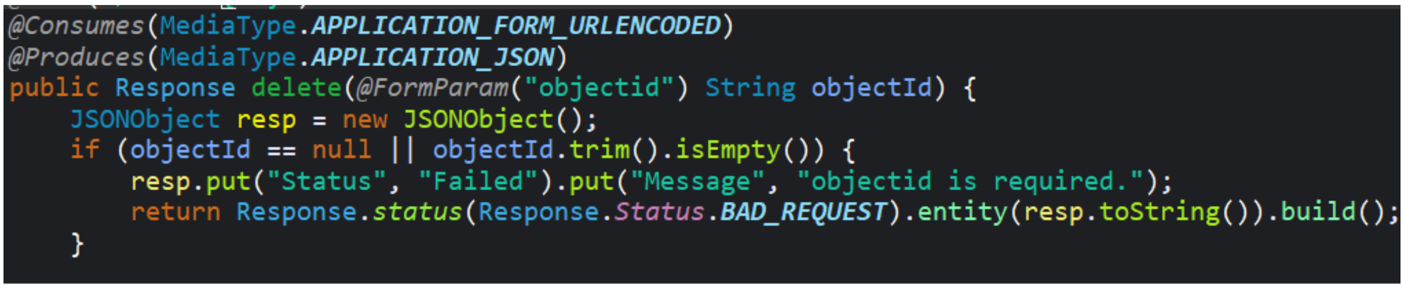
Purpose: Specifies the media types that the method/resource can **consume** from the HTTP request body.

Where to use: On a method or class.

Details:

- Indicates expected Content-Type of incoming request data.

Example:



```
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response delete(@FormParam("objectid") String objectId) {
    JSONObject resp = new JSONObject();
    if (objectId == null || objectId.trim().isEmpty()) {
        resp.put("Status", "Failed").put("Message", "objectid is required.");
        return Response.status(Response.Status.BAD_REQUEST).entity(resp.toString()).build();
    }
```

viii. **@PathParam**

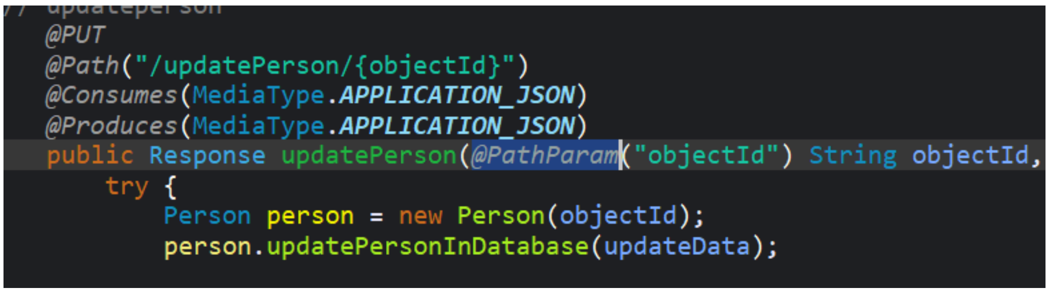
Purpose: Binds a method parameter to a URI path template variable.

Where to use: On method parameters**.**

Details:

- Extracts dynamic values from URI path segments.

Example:



```
// updateperson
@PUT
@Path("/updatePerson/{objectId}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response updatePerson(@PathParam("objectId") String objectId,
    try {
        Person person = new Person(objectId);
        person.updatePersonInDatabase(updateData);
```

ix. **@QueryParam**

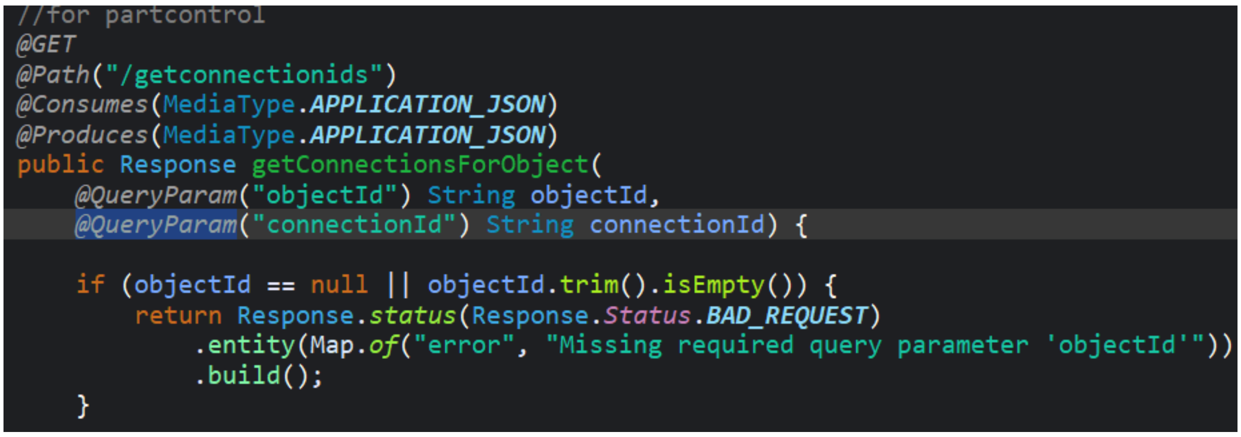
Purpose: Binds a method parameter to an HTTP query parameter.

Where to use: On method parameters.

Details:

- Extracts values from URL query strings, e.g/getconnectionids/objectId=0C95.18AF.6027.8A84.APN.

Example:



```
//for partcontrol
@GET
@Path("/getconnectionids")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response getConnectionsForObject(
    @QueryParam("objectId") String objectId,
    @QueryParam("connectionId") String connectionId) {

    if (objectId == null || objectId.trim().isEmpty()) {
        return Response.status(Response.Status.BAD_REQUEST)
            .entity(Map.of("error", "Missing required query parameter 'objectId'"))
            .build();
    }
```

x. @HeaderParam

Purpose: Binds a method parameter to an HTTP header value.

Where to use: On method parameters.

Details:

- Extracts HTTP header values.

Example:

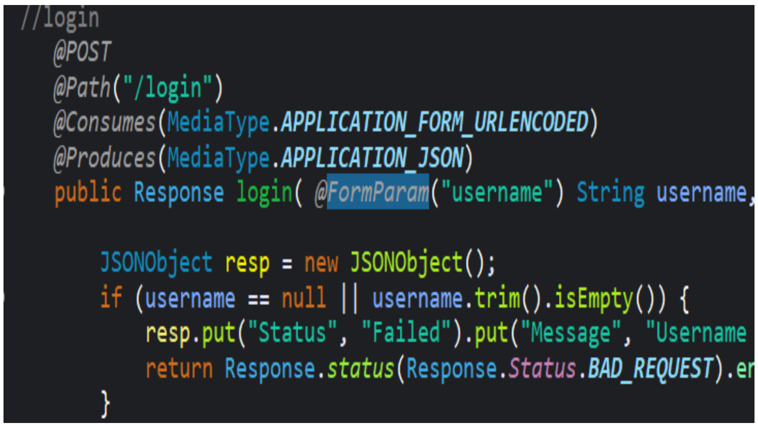
```
@GET
@Path("/resource")
public Response getResource(@HeaderParam("Authorization") String authHeader) {
   // Use authHeader for authentication/authorization
}
```

xi. @FormParam

Purpose: Binds a method Parameter to a form field value (for application/x-www-form-urlencoded POST requests).

Where to use? On a method.

Example:



```
//login
  @POST
  @Path("/login")
  @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
  @Produces(MediaType.APPLICATION_JSON)
  public Response login( @FormParam("username") String username,

      JSONObject resp = new JSONObject();
      if (username == null || username.trim().isEmpty()) {
          resp.put("Status", "Failed").put("Message", "Username
          return Response.status(Response.Status.BAD_REQUEST).er
      }
```

xii. @Context

Purpose: Injects contextual information into resource classes/methods.

Where to use: On method parameters or class fields.

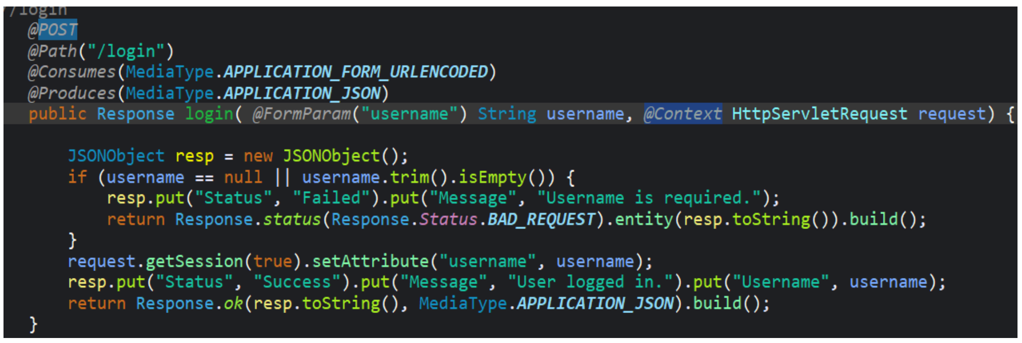
Details:

- Allows access to request info like URI details, HTTP headers, security context, etc.

Common injectable types:

- UriInfo — Information about URI details
- Request — The request context
- HttpHeaders — HTTP headers info
- SecurityContext — Security info of the request

Example:



```
//login
  @POST
  @Path("/login")
  @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
  @Produces(MediaType.APPLICATION_JSON)
  public Response login( @FormParam("username") String username, @Context HttpServletRequest request) {

      JSONObject resp = new JSONObject();
      if (username == null || username.trim().isEmpty()) {
          resp.put("Status", "Failed").put("Message", "Username is required.");
          return Response.status(Response.Status.BAD_REQUEST).entity(resp.toString()).build();
      }
      request.getSession(true).setAttribute("username", username);
      resp.put("Status", "Success").put("Message", "User logged in.").put("Username", username);
      return Response.ok(resp.toString(), MediaType.APPLICATION_JSON).build();
  }
```

9. CRUD OPERATIONS

9.1 Create (POST)

- Client sends data to create a new resource; server assigns an ID and returns the created resource and its URI.

9.2 Read (GET)

- Client requests data; server returns the resource(s) if found, or 404 if not.

9.3 Update (PUT)

- Client sends full updated data for a resource; server replaces existing data or returns 404 if the resource doesn’t exist.

9.4 Delete (DELETE)

- Client requests removal of a resource; server deletes it or returns 404 if not found.