**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

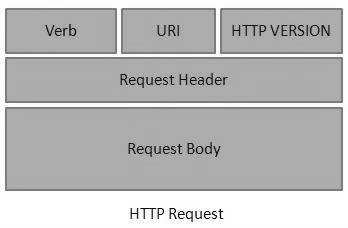
│ │ └── resources/

│ │ ├── application.properties <-- App config (DB, port, etc.)

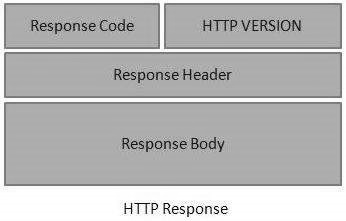
├── pom.xml <-- Maven configuration

**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**

* **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**

* 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.

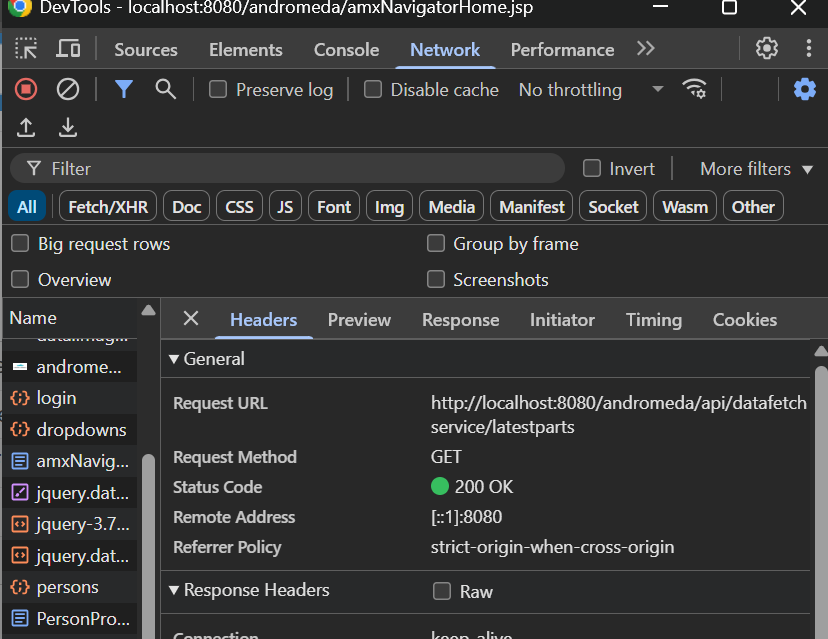
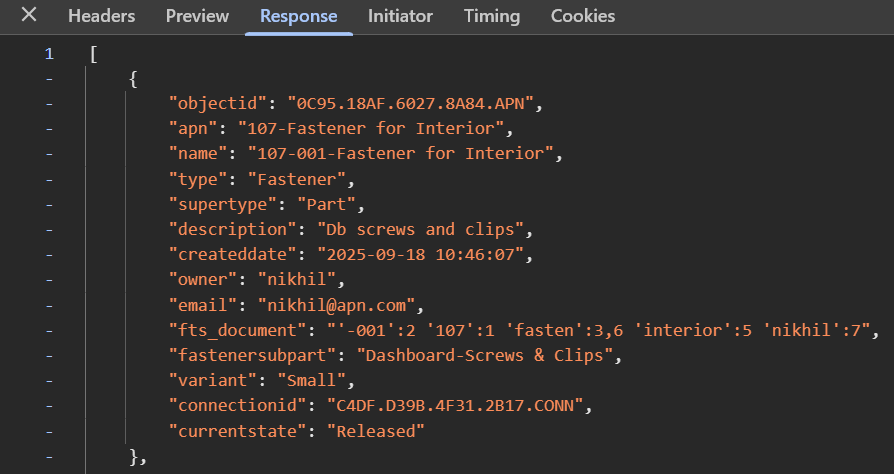
**6. REST Methods:**

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

**Example:**

**@**GET ---- annotation Response: Json Format

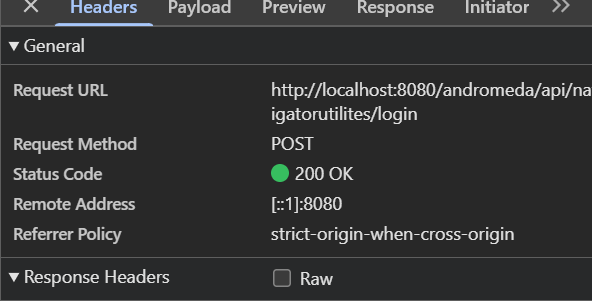
Request Method: GET

Accept: Application/Json

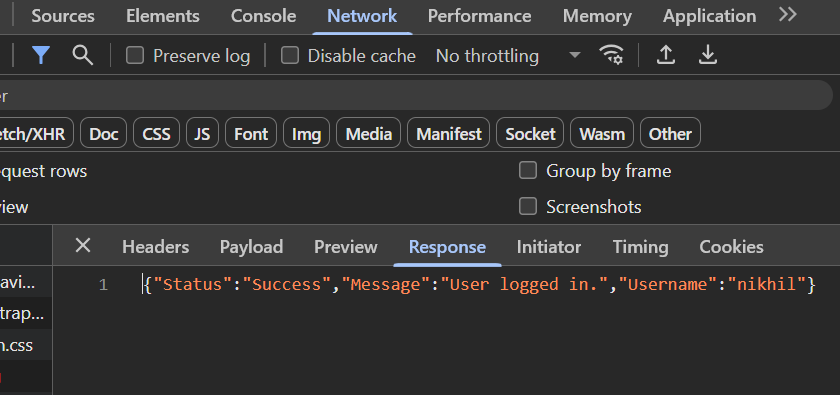
* **POST**
* **Purpose:** Create a new resource on the server
* **Request body:** Contains the data to create the resource
* **Response:** Returns the created message or often status code with a 201 created status and a locator header pointing to new resource.

**Example:**

**@**POST

Content-Type: Application/Json

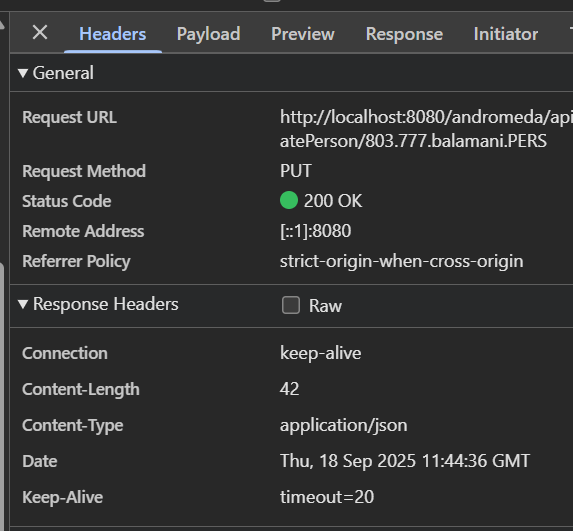
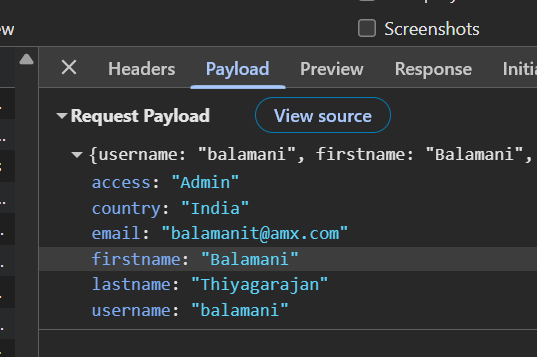
Response: Json Format

{"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 Request body:

Content-Type: Application/Json

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:

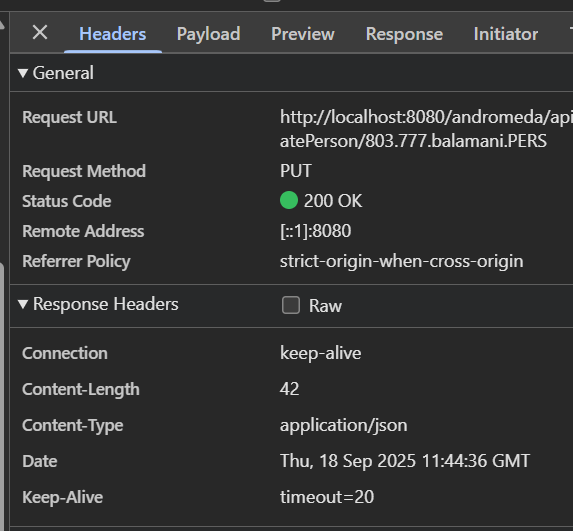
**7.1. 1xx – Informational Status code:**

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.

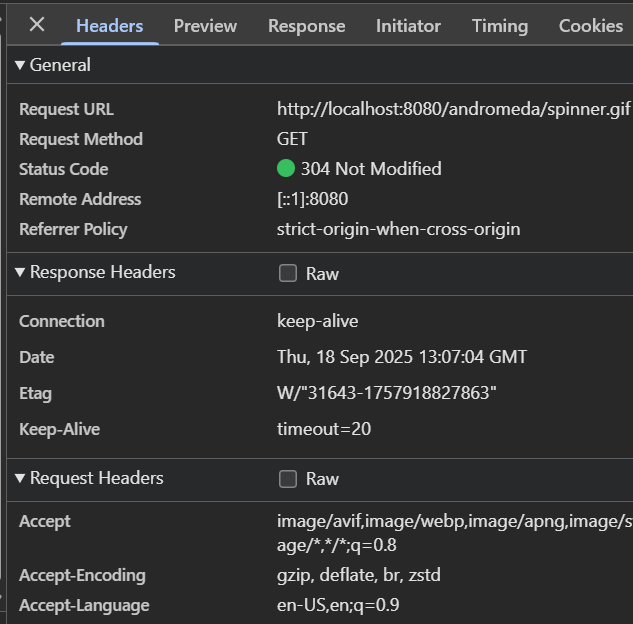
* **200 – OK:** Request succeeded, and response body contains the requested data (e.g., GET).
* **201 – Created:** A new resource was successfully created (e.g., POST). Location header should point to new resource URI**.**
* **202 – Accepted:** Request accepted but not yet processed (e.g., async operations).
* **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.

**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.

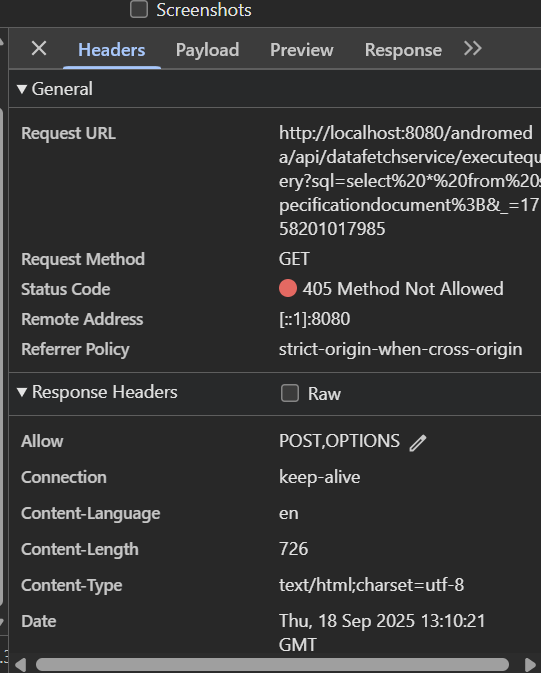
In REST APIs, these are rarely used directly by clients.

Example of Redirection Status code:

**7.4. 4xx – Client Errors Status code:**

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

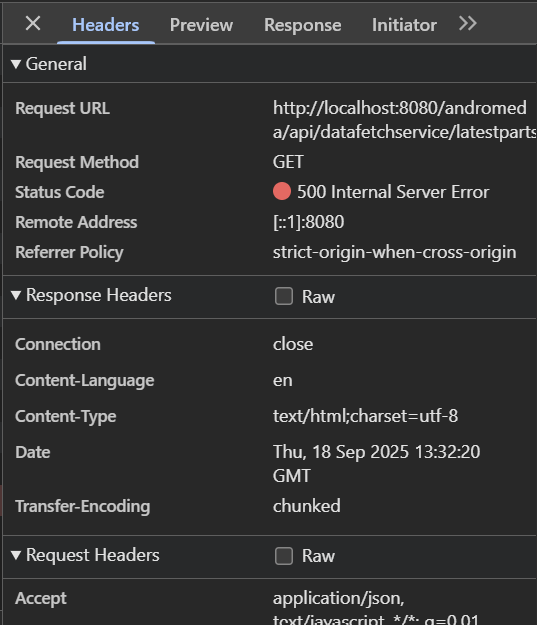
* **400–Bad Request:** The request is malformed or invalid (e.g., missing parameters, invalid JSON).
* **401–Unauthorized:** Authentication is required or has failed (e.g., missing or invalid token).
* **403–Forbidden:** The client is authenticated but does not have permission to access the resource.
* **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:

**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:

**8. JAX-RS REST Web Services Annotations**

1. **@PATH:**

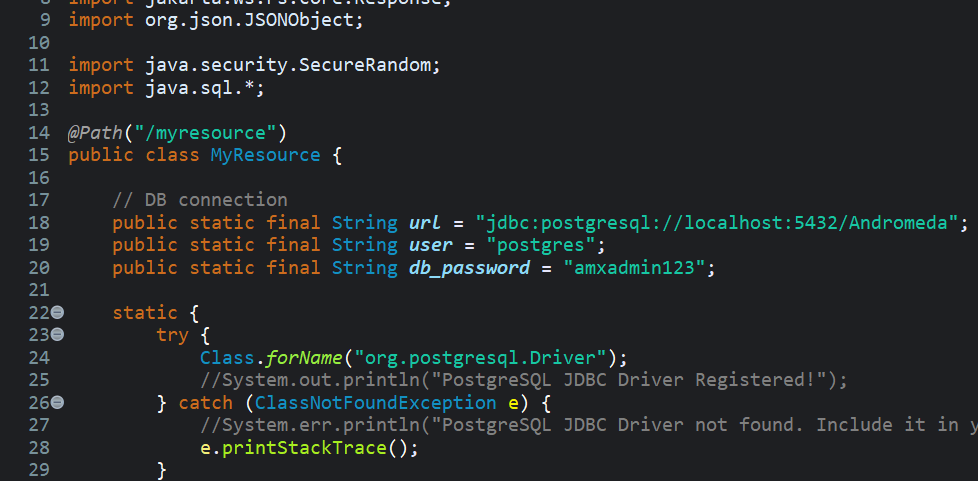
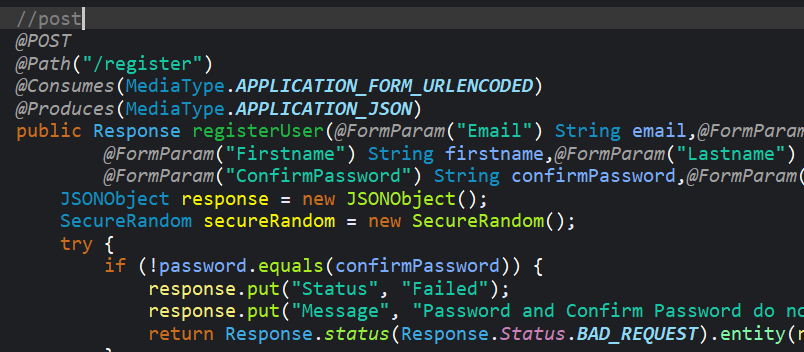
**Purpose:** Defines the relative URI path for a REST resource class or a specific method.

**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 2. On a method

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

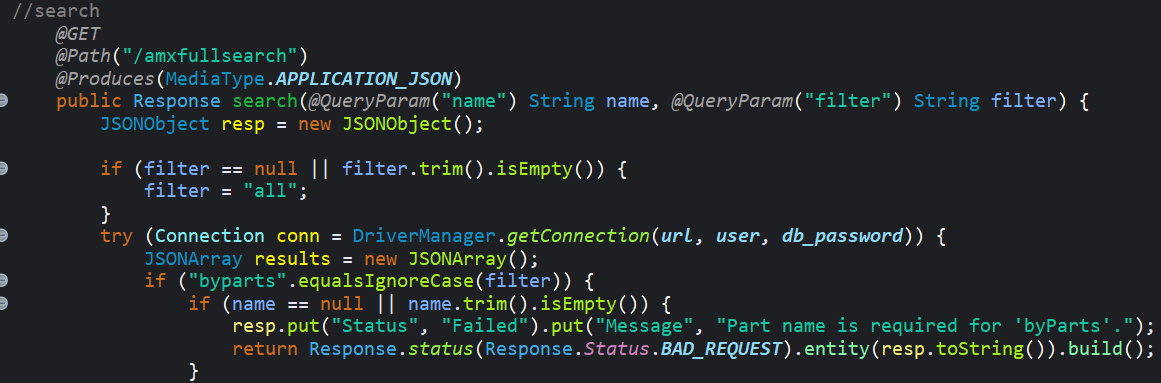
1. **@GET**

**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:**

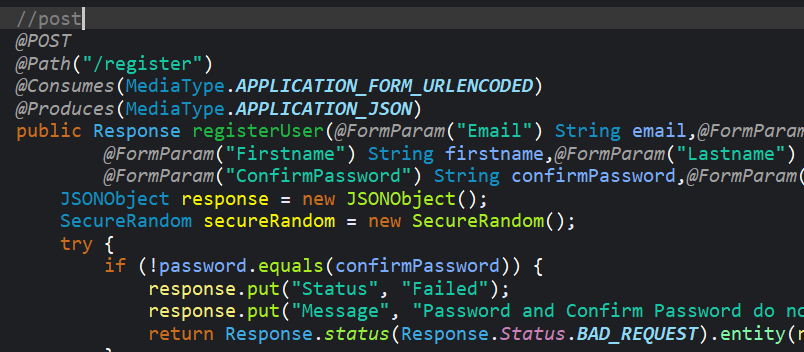
1. **@POST**

**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:**

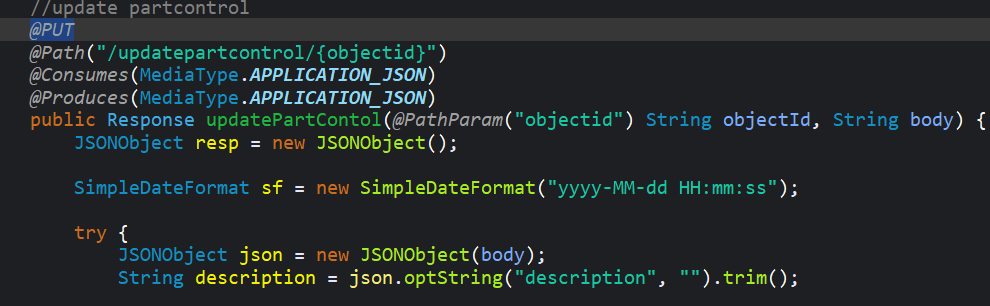
1. **@PUT**

**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:**

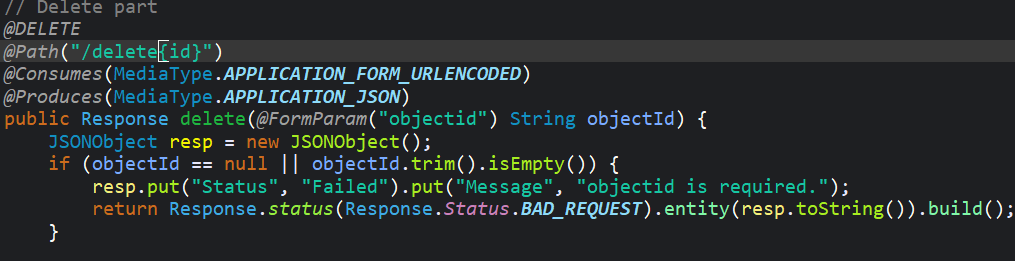
1. **@DELETE**

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

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

**Details:**

* Used to delete a resource.

**Example:**

1. **@Produces**

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

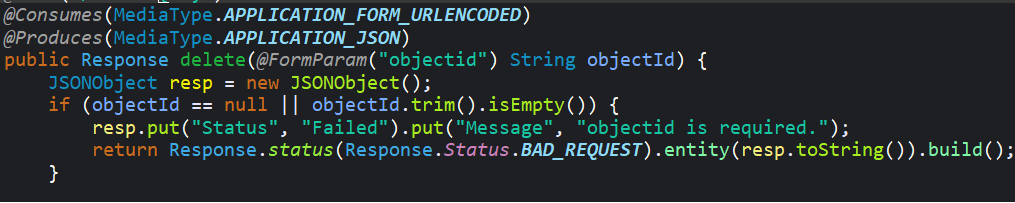
**Where to use:** On a method or class.

**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:**

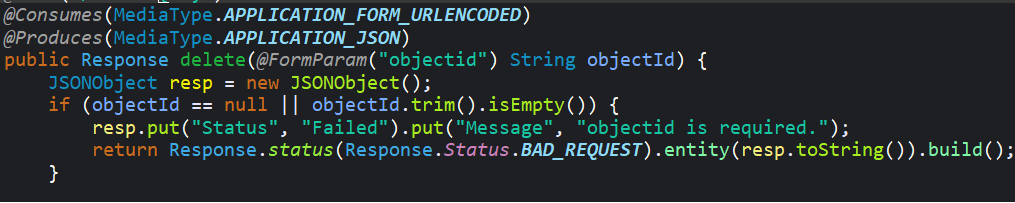
1. **@Consumes**

**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:**

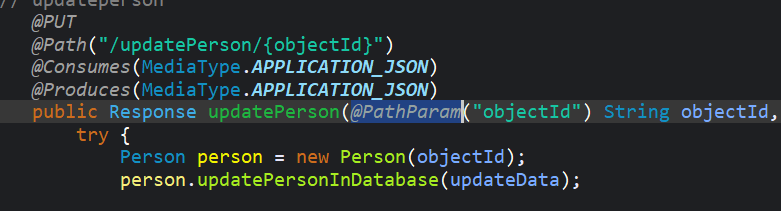
1. **@PathParam**

**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:**

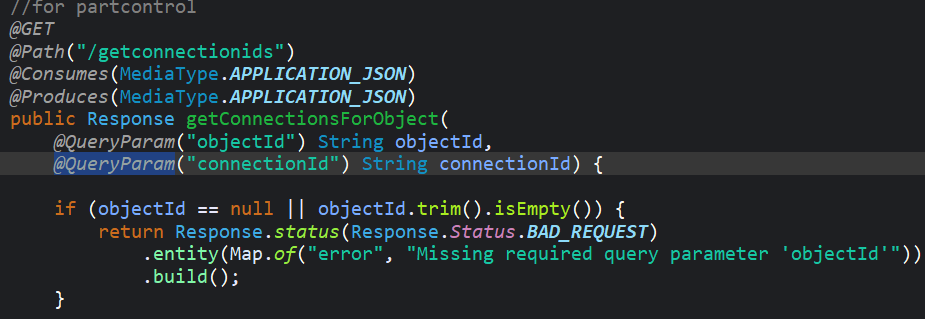
1. **@QueryParam**

**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:**

1. **@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) {

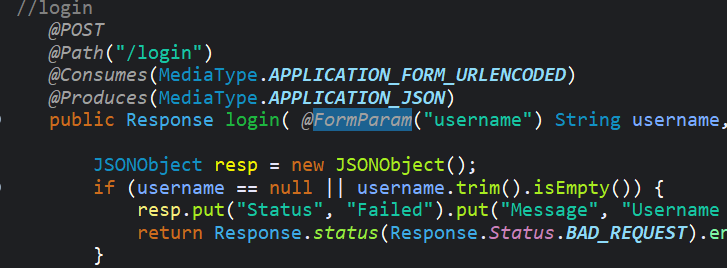
// Use authHeader for authentication/authorization

}

1. **@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:**

1. **@Context**

**Purpose:** Injects contextual information into resource classes/methods.

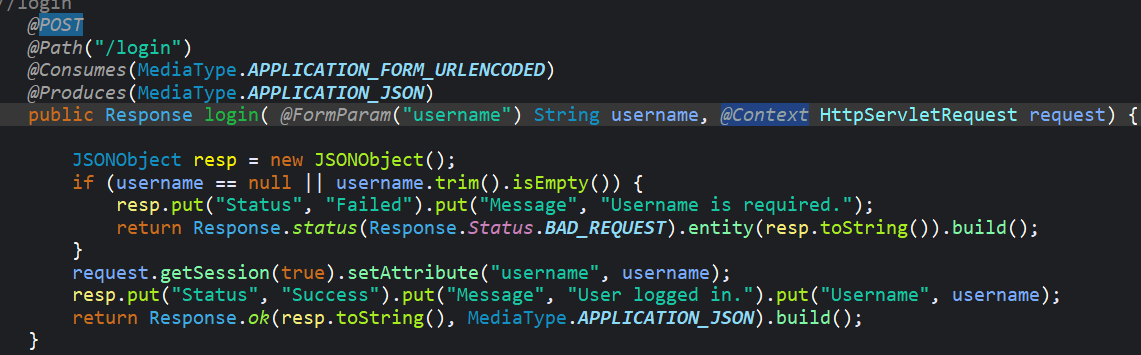
**Where to use:** On method parameters or class fields.

**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:**

**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.