Reference urls

|  |  |
| --- | --- |
| cxf | <https://cxf.apache.org/docs/jax-rs.html> |
| configuring jax-rs bean | <https://cxf.apache.org/docs/jaxrs-services-configuration.html> |
| spring boot | <https://cxf.apache.org/docs/springboot.html#SpringBoot-SpringBootCXFJAX-RSStarter>  <https://cxf.apache.org/docs/jaxrs-services-configuration.html#JAXRSServicesConfiguration-SpringBoot> |
| git hub | <https://github.com/apache/cxf/tree/main/distribution/src/main/release/samples/jax_rs/spring_boot_scan/application> |
|  |  |
|  |  |
| main cxf websites | <https://cxf.apache.org/>  <https://cxf.apache.org/docs/migration-guides.html> |
|  |  |

Current status-

unable to start app with switched to 21- but check if app is running or not

complete section 4 & 5 – to know about XSD file

solve that user name token program use a runner

RESTful webervices & advantages

JAX-RS – java api for xml based restful services

Same JAX-RS is the specification/API, and the implementation is given by restful service providers – apache cxf, jersey, REST Easy from JBoss (same like JPA spec)

Specification is nothing but set of rules and regulations, /api – it will just contains the interfaces, impl is from apache cxf

All the main annotations like @path, @Post …are part of javax.ws.rs

And this jax-rs internally uses http protocol like GET,PUT, POST… (means clients should use these protocols to send the data and server will return these reponses 200,302)

Apache CXF –By default supports xml, anything other than that needs to be mentioned specifically, @POST(produces= Application/JSON..)

Means by default while incoming, the xml payload will be binded to java object and while returning the response, java object will be converted into xml payload

To build web services there are 2 ways- soap and REST

Rest means- representational state transfer

REST was defined by Roy Fielding in his 2000 doctoral dissertation.

### 1. What is REST?

RESTful web services heavily uses HTTP protocol

While REST often uses HTTP, it's not strictly tied to it. You could theoretically implement a RESTful system over other protocols (though HTTP is by far the most common and natural fit for its principles)

REST is an architectural style, not a protocol or a standard in itself.

Its like API It's a set of guidelines and constraints and best practices for designing distributed systems, when applied, result in a system that is often:

Unlike soap , It doesn't define the syntax of messages or the exact protocols, but rather the overall design philosophy where as soap tell the msg should be like soap envelope, body , faults..

* **Scalable:** Easy to extend and handle more requests.
* **Simple:** Easier to understand and implement.
* **Modifiable:** Changes in one part of the system have minimal impact on others.
* **Portable:** Can be deployed across various platforms.

### 2. Key Principles (Constraints) of REST

To be considered "RESTful," a web service should adhere to the following six architectural constraints:

* **Client-Server Architecture:**
  + The client (e.g., a web browser, mobile app) and the server (where the service is hosted) are separate and independent.
  + This separation of concerns improves portability and scalability. The client doesn't need to know about the server's internal logic, and the server doesn't need to know about the client's UI.
* **Statelessness:**
  + This means the server doesn't "remember" previous interactions with a specific client. every request is fresh request

This simplifies server design, improves scalability (as any server can handle any request), and makes the service more robust to failures.

* + Each request from a client to a server must contain all the information necessary to understand the request. The server should not store any client context between requests. Ex:- client should send the jwt token / should keep crede in header in every req,
* **Cacheability:**
  + Responses from the server should explicitly or implicitly define themselves as cacheable or non-cacheable.
  + This allows clients or intermediaries (like proxies) to cache responses, improving performance and reducing server load for subsequent requests for the same resource.
* **Layered System:**
  + A client typically cannot tell whether it's connected directly to the end server or to an intermediary server along the way.
  + This allows for the use of proxy servers, load balancers, and other intermediaries to enhance scalability, security, and performance without affecting the client or the end server.
* **Uniform Interface:** This is the most crucial constraint and defines how the client and server interact. It consists of four sub-principles:
  + **Identification of Resources:** Individual resources are identified in requests, e.g., using URLs (/users/123, /products).
  + **Manipulation of Resources Through Representations:** Clients manipulate resources by exchanging representations of those resources (e.g., JSON or XML). For instance, to create a user, you send a JSON representation of the user.
  + **Self-Descriptive Messages:** Each message includes enough information to describe how to process it. This means the server can indicate the media type of the response (e.g., Content-Type: application/json) or the client can specify the desired media type (e.g., Accept: application/json).
  + **Hypermedia as the Engine of Application State (HATEOAS):** This is often considered the most difficult constraint to fully implement. It means that responses should include links to related resources, allowing the client to discover available actions and transitions. For example, a response for a "user" resource might include links to their "orders" or an "update user" endpoint.
* **Code on Demand (Optional):**
  + Servers can temporarily extend or customize client functionality by transferring executable code (e.g., JavaScript). This constraint is optional and less commonly used in typical REST APIs.

### 3. How do RESTful Web Services Work?

RESTful services primarily use the standard HTTP methods to perform operations on resources:

* **GET:** Retrieves a representation of a resource. (Idempotent and safe)
  + Example: GET /users/123 (get user with ID 123)
* **POST:** Creates a new resource or submits data to be processed. (Not idempotent)
  + Example: POST /users (create a new user)
* **PUT:** Updates an existing resource (or creates it if it doesn't exist, replacing the entire resource). (Idempotent)
  + Example: PUT /users/123 (update user with ID 123, replacing all its data)
* **DELETE:** Deletes a resource. (Idempotent)
  + Example: DELETE /users/123 (delete user with ID 123)
* **PATCH:** Partially updates an existing resource. (Not idempotent, but generally preferred for partial updates over PUT)
  + Example: PATCH /users/123 (update only specific fields of user with ID 123)

**Resources** are the core concept. Anything that can be named and accessed is a resource. Examples: users, products, orders, blog posts.

### 4. Advantages of RESTful Web Services

* **Flexibility:** Supports various data formats (JSON, XML, plain text, etc.).,but in soap we must use soap message format With xml only
* **Simplicity:** Easier to build and consume compared to older styles like SOAP. And since REST supports json , xml parsing overhead will be decreased, and payload size will be decreased
* **Scalability:** Statelessness and caching contribute to better scalability.
* **Decoupling:** Client and server are independent, allowing for separate development and deployment.
* **Standard HTTP:** Leverages existing web infrastructure and tools.
* **Interoperability/ Re-usability:**- a python app/.Net app / any heterogeneous app can talk to java app using HTTP REST , both SOAP and REST webservices are re-usable

**Note:**- caching can be implemented in both rest & SOAP, in SOAP we have a handler class(filter) which will be invoked for every request, so check for duplicate req and u can return the data from the filter itself..

### 5. REST in the Context of Spring Data JPA and Java

When you're working with Spring Data JPA and Java, you'll typically expose your JPA entities (which represent your data) as **RESTful resources** using **Spring Boot** and **Spring Web (Spring MVC)**.

* You'll define **Controller** classes in Spring that map HTTP requests (like GET, POST, PUT, DELETE) to specific methods.
* These methods will then interact with your **Spring Data JPA repositories** to perform CRUD (Create, Read, Update, Delete) operations on your entities (e.g., userRepository.findById(id), userRepository.save(user)).
* The data exchanged between the client and server will typically be in **JSON format**, which Spring handles automatically for serialization and deserialization.

**Example (simplified Spring Boot REST Controller):**

Java

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController // Marks this class as a REST Controller

@RequestMapping("/api/users") // Base path for user-related endpoints

public class UserController {

private final UserRepository userRepository; // Your Spring Data JPA repository

public UserController(UserRepository userRepository) {

this.userRepository = userRepository;

}

@GetMapping // Handles GET /api/users

public List<User> getAllUsers() {

return userRepository.findAll();

}

@GetMapping("/{id}") // Handles GET /api/users/{id}

public User getUserById(@PathVariable Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new RuntimeException("User not found"));

}

@PostMapping // Handles POST /api/users

public User createUser(@RequestBody User user) { // @RequestBody maps JSON to User object

return userRepository.save(user);

}

@PutMapping("/{id}") // Handles PUT /api/users/{id}

public User updateUser(@PathVariable Long id, @RequestBody User userDetails) {

User user = userRepository.findById(id)

.orElseThrow(() -> new RuntimeException("User not found"));

user.setName(userDetails.getName());

user.setEmail(userDetails.getEmail());

// ... update other fields

return userRepository.save(user);

}

@DeleteMapping("/{id}") // Handles DELETE /api/users/{id}

public void deleteUser(@PathVariable Long id) {

userRepository.deleteById(id);

}

}

This example demonstrates how Spring Boot simplifies the creation of RESTful web services that interact with your data layer (managed by Spring Data JPA). Understanding REST is fundamental for building modern, distributed Java applications

Architecture

JAX-RS provides exception mappers

- to map custom java exceptions to error codes- ex:- when my desired exception came, then It should give 400.. some status for that runtime exception

Spring REST vs JAX-RS

**Not a JAX-RS Implementation:** This is a key difference: **Spring REST does not implement the JAX-RS specification.** It provides its own alternative way of building REST services in Java.

JAX-RS implementation is given only by Apache cxf, REST easy, jersey

Annotations

By default cxf framework supports xml, anything other than that needs to be mentioned specifically, @POST(produces= Application/JSON..)

@Path, @GET, @POST, @PUT, @DELETE, @Produces, @Consumes, @PathParam, @QueryParam

|  |  |
| --- | --- |
| @path(“/users/{username”}) | this anno is used to bind the url to a java method  this anno can be used on both class and method |
| @PathParam | to fetch the values from the url  ex:- |
| @QueryParam | to fetch the values from the url after question mark “?” |
| @FormParam | used to map the params the comes during a form submission |
| ~~@RequestBody~~ | To map the payload to a java object like in REST we don’t have any @RequestBody,  here that full payload will be binded automatically to that java object |
| @Provider | to map custom java exceptions to error codes  import javax.ws.rs.core.Response;  import javax.ws.rs.ext.ExceptionMapper;  import javax.ws.rs.ext.Provider;  @Provider // Important: tells JAX-RS runtime to discover this mapper  public class UserNotFoundExceptionMapper implements ExceptionMapper<UserNotFoundException> { |
| @Consumes  @Produces |  |

Exception mapper

|  |  |
| --- | --- |
| so here, when ever this is UserNotFoundException thrown then automatically 400 response will be thrown with the error message object   * This Exception mapper is same like in REST controller we have the controller advice,   // Custom Error Response POJO (example)  class ErrorResponse {  private int status;  private String errorCode;  private String message;  public ErrorResponse(int status, String errorCode, String message) {  this.status = status;  this.errorCode = errorCode;  this.message = message;  }  // Getters for JSON serialization  public int getStatus() { return status; }  public String getErrorCode() { return errorCode; }  public String getMessage() { return message; }  }  // Custom Exception (example)  class UserNotFoundException extends RuntimeException {  public UserNotFoundException(String message) {  super(message);  }  } | import javax.ws.rs.core.Response;  import javax.ws.rs.ext.ExceptionMapper;  import javax.ws.rs.ext.Provider;  @Provider // Important: tells JAX-RS runtime to discover this mapper  public class UserNotFoundExceptionMapper implements ExceptionMapper<UserNotFoundException> {  @Override  public Response toResponse(UserNotFoundException exception) {  // Build a custom error response  ErrorResponse error = new ErrorResponse(  404, // Custom error code  "NOT\_FOUND",  exception.getMessage() // Get the message from the exception  );  return Response.status(Response.Status.NOT\_FOUND) // Set HTTP 404 status  .entity(error) // Set the custom error object as the response body  .type("application/json") // Specify content type  .build();  }  } |

|  |  |
| --- | --- |
| usage  @GET  @Path("/{id}")  @Produces(MediaType.APPLICATION\_JSON)  public User getUser(@PathParam("id") Long id) {  User user = userService.findUserById(id); // Assume this throws UserNotFoundException if not found  if (user == null) {  throw new UserNotFoundException("User with ID " + id + " not found.");  }  return user;  } | client receives below in postman  HTTP/1.1 404 Not Found  Content-Type: application/json  {  "status": 404,  "errorCode": "NOT\_FOUND",  "message": "User with ID 123 not found."  } |

Read in gemini on “is spring REST and jax-rs are both same?”

Code

<https://cxf.apache.org/docs/springboot.html#SpringBoot-SpringBootCXFJAX-RSStarter>

Maven jars

|  |  |
| --- | --- |
| for spring boot 3.2.2 u can use cxf- spring 4.0.0 | jakarta.ws.rs-api  this is the main jar, which will come automatically from spring boot starter |
| <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-spring-boot-starter-jaxrs</artifactId>  <version>4.0.0</version>  </dependency> | <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>3.2.2</version>  <relativePath /> <!-- lookup parent from repository -->  </parent> |
| <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.jakarta.rs/jackson-jakarta-rs-json-provider -->  To send and receive data as json use this jar  <dependency>  <groupId>com.fasterxml.jackson.jakarta.rs</groupId>  <artifactId>jackson-jakarta-rs-json-provider</artifactId>  <version>2.18.4</version>  </dependency> |  |

Spring props

<https://cxf.apache.org/docs/springboot.html#SpringBoot-SpringBootCXFJAX-RSStarter>

Use "**cxf.path**" property to customize a CXFServlet URL pattern

Use "**cxf.servlet.init**" map property to customize CXFServlet properties such as "services-list-path" (available by default at  "/services"), etc.

Use "**cxf.servlet.loadOnStartup**" set loadOnStartup priority of the CXFServlet (by default, -1)

Use "**cxf.servlet.enabled**" enable/disable CXFServlet regsitration (since **3.3.12**/**3.4.5**[)](https://issues.apache.org/jira/issues/?jql=project+%3D+CXF+AND+fixVersion+%3D+3.5.0)

Use "**cxf.jaxrs.component-scan**" property to create a JAX-RS endpoint from the auto-discovered JAX-RS root resources and providers which are marked as Spring Components (annotated with Spring @Component or created and returned from @Bean methods).

Use "**cxf.jaxrs.component-scan**-**packages**" property to restrict which of the auto-discovered Spring components are accepted as JAX-RS resource or provider classes. It sets a comma-separated list of the packages that a given bean instance's class must be in. Note, this property, if set, is only effective if a given bean is a singleton. It can be used alongside or as an alternative to the "**cxf.jaxrs.component-scan**-**beans**" property. This property is available starting from CXF 3.1.11.

Server.context-path= /restws to setup a context root for an app

Assume u annotate the classes with @Path , in spring REST and spring boot these classes will be auto recognized, and we can directly hit from postman

but now here if u want spring to auto recognize our classes annotated with @path we have keep below

|  |  |
| --- | --- |
| The below properties are only for json data transmission  cxf.jaxrs.classes-scan=true  cxf.jaxrs.classes-scan-packages=com.fasterxml.jackson.jakarta.rs,com.bharath.restws  note:- try without these props and see what error will come |  |

HTTP status

304 means not modified- ex:- when user tried to delete and if there is no data and if u didn’t delete, then give as 304 not modified

CRUD program

By default cxf framework supports xml, so if u are sending an receiving xml data, then no need to mention as xml anywhere

Misc api

|  |  |
| --- | --- |
| @Context  Uri uri | if u want to see the load balancer url /ip address of ur current app where it is hosted then use this class |
|  |  |

Sending data as json

Add json dependency

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.jakarta.rs/jackson-jakarta-rs-json-provider -->

<dependency>

<groupId>com.fasterxml.jackson.jakarta.rs</groupId>

<artifactId>jackson-jakarta-rs-json-provider</artifactId>

<version>2.18.4</version>

</dependency>

The below properties are only for json

cxf.jaxrs.classes-scan=true

cxf.jaxrs.classes-scan-packages=com.fasterxml.jackson.jakarta.rs,com.bharath.restws (it should scan these because it should find the jackson provider automatically)

because the provider is responsible to convert the java object to json data vice versa and give our base package where our @Produces(app/json) is present