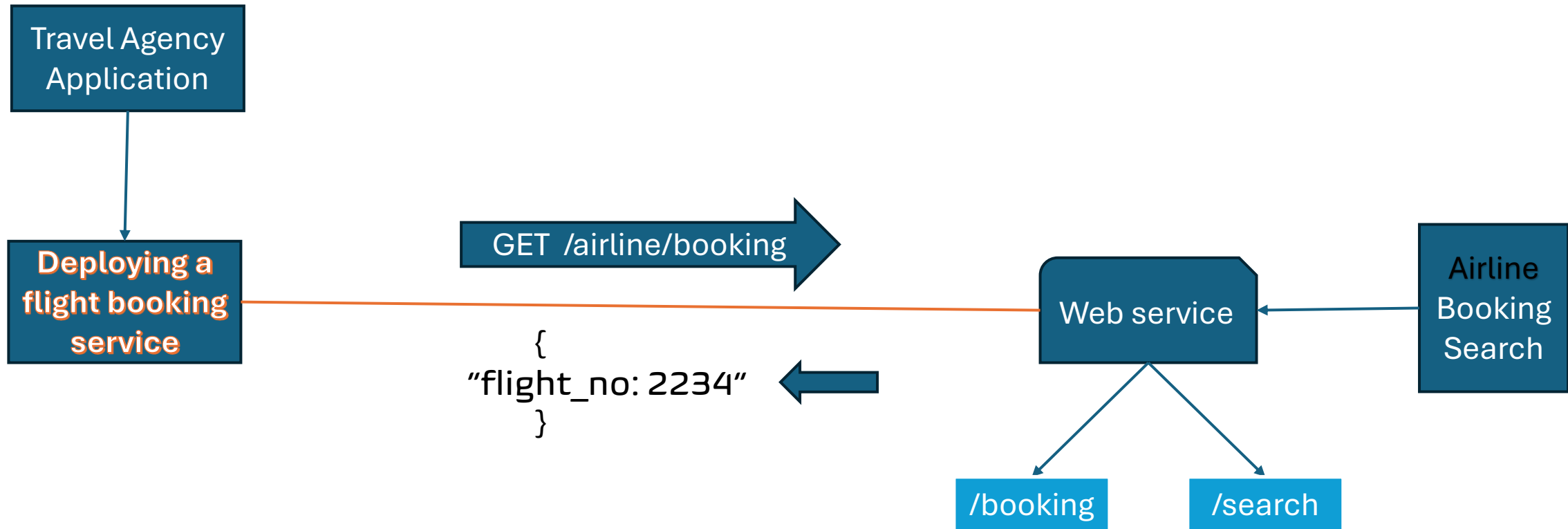


5COSC022W

Lecture Week 05

REST API (JAX-RS)

Dr. Hamed Hamzeh

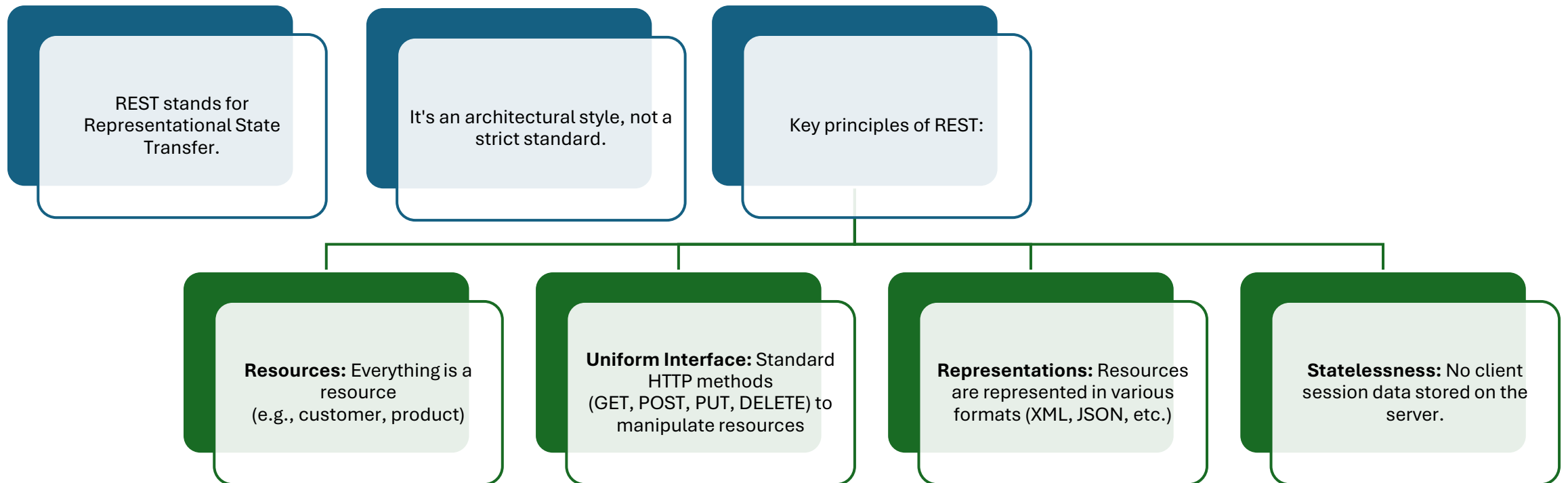


Question!

What kind of web service is suitable for this architecture?

RESTful service

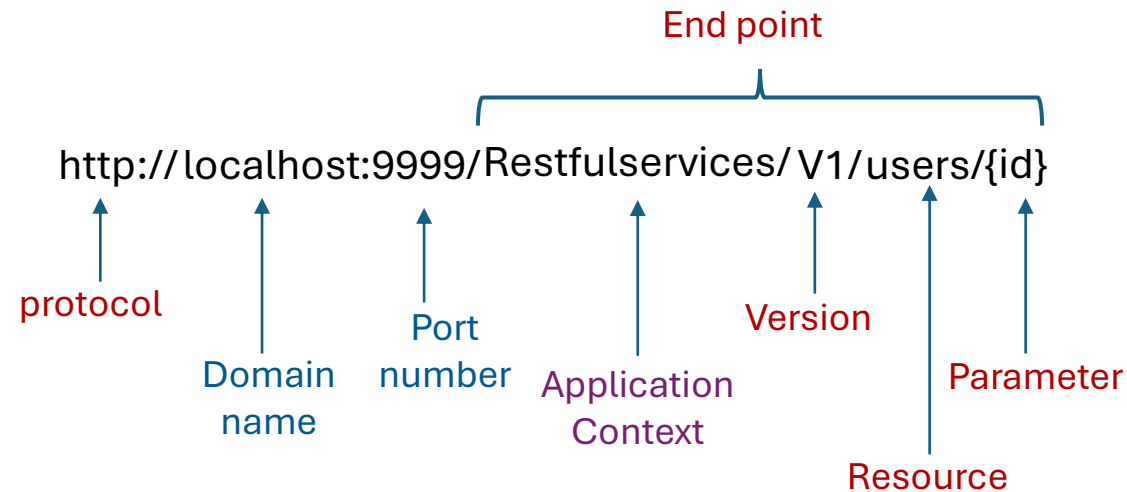
What is REST?



Defining a RESTful resource

A REST resource can be defined as an object that is of a specific type with the associated data and is optionally associated to other resources.

We can do the HTTP requests against a resource such as the [HEAD](#), [GET](#), [POST](#), [PUT](#), and [DELETE](#) methods.



What is JAX-RS?

Definition:

- JAX-RS, or Java API for RESTful Web Services, is a set of Java APIs that facilitate the development of RESTful web services.

Integration:

- Originally part of Java EE (Enterprise Edition), it is now a key component of [Jakarta EE](#), emphasizing its role in enterprise-level Java applications.

Simplifying Development:

- JAX-RS simplifies the creation of RESTful services, allowing developers to focus on building scalable and interoperable web services without getting bogged down by low-level details.

Key Features of JAX-RS

Annotation-driven:

- Use annotations to define resources, methods, and parameters.

HTTP Methods:

- Support for common HTTP methods - GET, POST, PUT, DELETE, etc.

URI Mapping:

- Map URIs to resource classes and methods.

Content Negotiation:

- Handle different data formats (XML, JSON, etc.).

Exception Handling:

- Graceful handling of exceptions.

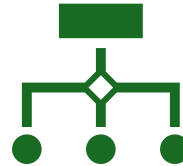
Resource Classes



Java Classes as Web Resources:

In JAX-RS, a resource class is a Java class that models a web resource.

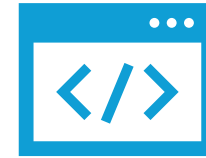
It encapsulates the functionality of a specific resource or a collection of related resources.



@Path Annotation:

The @Path annotation is used to associate a resource class or method with a URI path.

It defines the base URI for the resource, making it accessible via HTTP.



HTTP Methods and Annotations:

Resource methods within the class are annotated with HTTP methods like @GET, @POST, @PUT, etc.

These annotations define the type of HTTP request the method can handle.

Jersey



Jersey is the official reference implementation of JAX-RS.



Developed by Sun Microsystems (now Oracle) and maintained by the Eclipse Foundation.



Provides a standard and consistent approach to building RESTful services.



Provides a comprehensive set of APIs that implement JAX-RS, simplifying the development of RESTful services in Java.



Leverages annotations to define resources, HTTP methods, and other aspects



Well-documented



Jersey integrates seamlessly with Java EE (Enterprise Edition) and Jakarta EE, making it suitable for enterprise-level applications.

Annotations in JAX-RS

@Path

- Defines the base URI for the resource class or method.

@GET, @POST, @PUT, @DELETE

- Specifies the HTTP method for the resource method.

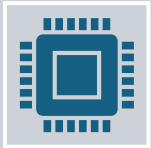
@PathParam

- Binds method parameters to a path segment.

@Produces, @Consumes

- Specify the media types the resource can produce or consume.

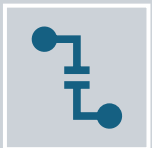
Dependencies



Dependencies refer to external libraries or modules that your project relies on to implement the functionality provided by the JAX-RS API.



These dependencies are typically required to compile, run, and deploy JAX-RS applications.



The most common dependency for JAX-RS applications is the implementation of the JAX-RS specification itself, and Jersey is a popular choice as the reference implementation.

Jersey Dependency:

- If you choose Jersey as the implementation for JAX-RS in your project, you need to include the Jersey libraries as dependencies.
- For Maven projects, you can add the following dependency in your pom.xml:

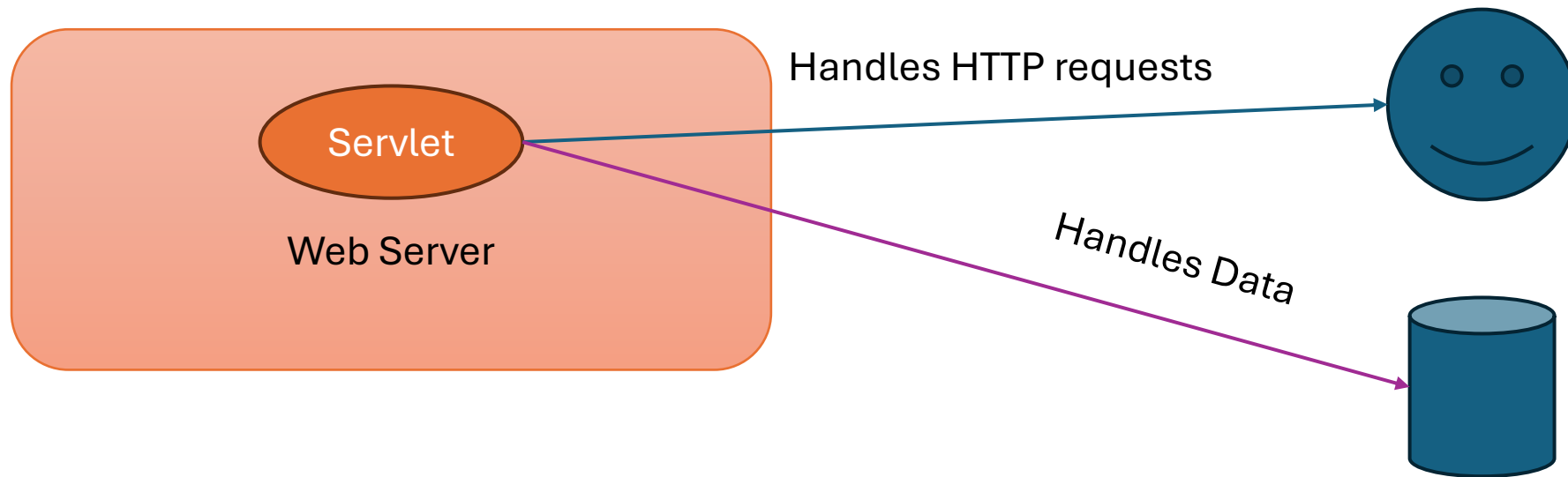
```
<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>2.35</version> <!-- Use the latest version -->
  </dependency>
</dependencies>
```

Additional Dependencies

If you are deploying your JAX-RS application as a servlet, you'll need a servlet container (e.g., Apache Tomcat, Jetty) as a dependency.

This is necessary for hosting and running your JAX-RS application.

Servlet



servlet



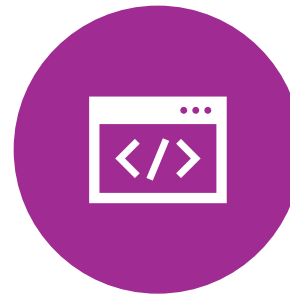
Servlets are Java classes that extend the capabilities of a server. They are part of the Java EE or Jakarta EE platform.



Have well-defined lifecycle. Web server initialises servlet.



Servlets process HTTP requests and generate HTTP responses.



It supports multi-threading

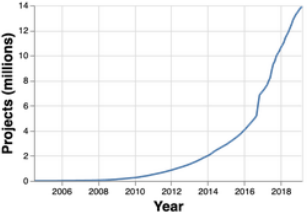
How to get the plugins and dependencies

MVN REPOSITORY

Search for groups, artifacts, categories

Search


Indexed Artifacts (32.6M)



Popular Categories


- Testing Frameworks & Tools
- Android Packages
- Logging Frameworks
- Java Specifications
- JSON Libraries
- Core Utilities
- JVM Languages
- Mocking

What's New in Maven




Scala Library
[org.scala-lang](#) » [scala-library](#) » 2.12.18-M1
Standard library for the Scala Programming Language
Last Release on Oct 8, 2022

33,563 usages
Apache



Scala Reflect
[org.scala-lang](#) » [scala-reflect](#) » 2.12.18-M1
Reflection Library for the Scala Programming Language
Last Release on Oct 8, 2022

3,900 usages
Apache



Scala Compiler
[org.scala-lang](#) » [scala-compiler](#) » 2.12.18-M1
Compiler for the Scala Programming Language
Last Release on Oct 8, 2022

2,116 usages
Apache

JAX-RS annotations- Path

- The **@javax.ws.rs.Path** annotation indicates the URI path to which a resource class or a class method will respond.
- The value that you specify for the @Path annotation is relative to the URI of the server where the REST resource is hosted.
- This annotation can be applied at both the class and the method levels.
- The following code snippet illustrates how you can make a class respond to a URI path template containing the /departments path fragment:

```
import javax.ws.rs.Path;

@Path("departments")
public class DepartmentService {
    //Rest of the code goes here
}
```


JAX-RS annotations- Path

- For an annotated method, the base URI is the effective URI of the containing class.
- For instance, you will use the URI of the following form to invoke the `getTotalDepartments()` method defined in the `DepartmentService` class:

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
@Path("departments")
public class DepartmentService {
    @GET
    @Path("count")
    @Produces("text/plain")
    public Integer getTotalDepartments() {
        return findTotalRecordCount();
    }
    //Rest of the code goes here
}
```

Specifying variables in the URI path template

The URI path template allows you to define variables that appear as placeholders in the URI. These variables would be replaced at runtime with the values set by the client.

The URI path template looks like `/departments/{id}`.

At runtime, the client can pass an appropriate value for the `id` parameter to get the desired resource from the server.

For instance, the URI path of the `/departments/10` format returns the IT department details to the caller.

```
import javax.ws.rs.Path;
import javax.ws.rs.DELETE;

@Path("departments")
public class DepartmentService {

    @DELETE
    @Path("{id}")
    public void removeDepartment(@PathParam("id")
    short id) {
        removeDepartmentEntity(id);
    }
    //Other methods removed for brevity
}
```

→ Specifying variable in URI path

Restricting values for path variables with regular expressions

AX-RS lets you use regular expressions in the URI path template for restricting the values set for the path variables at runtime by the client.

For example, you can set the regular expression as given in the following code snippet in order to ensure that the department name variable present in the URI path consists only of lowercase and uppercase alphanumeric characters:

```
@DELETE
@Path("/{name: [a-zA-Z][a-zA-Z_0-9]}")
public void removeDepartmentByName(@PathParam("name")
    String deptName) {
    //Method implementation goes here
}
```

Strict the resource name to
upper case and lowercase

Q: What happens if a specified name does not match with the regular expression?

Answer: the system reports the status back to the caller with an appropriate HTTP status code, such as 404 Not Found, which tells the caller that the requested resource could not be found at this moment.

Media Types

Media types define the format of the data being transmitted between the client and server in RESTful web services.

They enable the client and server to understand the structure and encoding of the data being exchanged.

Media types can be specified using the `@Produces` and `@Consumes` annotations in JAX-RS resource classes and methods.

`@Produces`

- is used to specify the media types that the resource method can produce (i.e., send back to the client).

`@Consumes`

- is used to specify the media types that the resource method can consume (i.e., accept from the client).

Example Media Types

`application/json:`

- Represents JSON (JavaScript Object Notation) data format.

`application/xml:`

- Represents XML (Extensible Markup Language) data format.

`text/plain:`

- Represents plain text data format.

`multipart/form-data:`

- Represents form data as submitted in HTML forms.

`application/x-www-form-urlencoded:`

- Represents form data encoded as a query string.

Negotiation and Content-Type Header



During communication, the client and server negotiate the media type to be used based on the **Accept** and **Content-Type** headers.



The Accept header in the client request specifies the media types that the client is willing to accept.



The Content-Type header in the request payload specifies the media type of the data being sent by the client.

@Produces Annotation

- The following example uses the `@Produces` annotation at the class level in order to set the default response media type as JSON for all resource methods in this class.
- At runtime, the binding provider will convert the Java representation of the return value to the JSON format.

```
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;
```

Import necessary requirements

```
@Path("departments")  
@Produces(MediaType.APPLICATION_JSON)  
public class DepartmentService{  
    //Class implementation goes here...  
}
```

We use media/content
type as JSON format

@Consumes Annotation

- The following example illustrates how you can use the @Consumes attribute to designate a method in a class to consume a payload presented in the JSON media type.
- The binding provider will copy the JSON representation of an input message to the Department parameter of the `createDepartment()` method.

```
import javax.ws.rs.Consumes;  
import javax.ws.rs.core.MediaType;  
import javax.ws.rs.POST;
```

```
@POST
```

```
@Consumes(MediaType.APPLICATION_JSON)
```

```
public void createDepartment(Department entity) {  
    //Method implementation goes here...  
}
```

Specifying the media type for the consumer

Annotations for processing HTTP request methods - GET

- A RESTful system uses the HTTP GET method type for retrieving the resources referenced in the URI path.
- The `@javax.ws.rs.GET` annotation designates a method of a resource class to respond to the [HTTP GET requests](#).
- In the following example, the REST URI for accessing the `findAllDepartments()` method may look like [/departments](#). The complete URI path may take the following URI pattern:

Specify the Path
Which
represents the
base URI

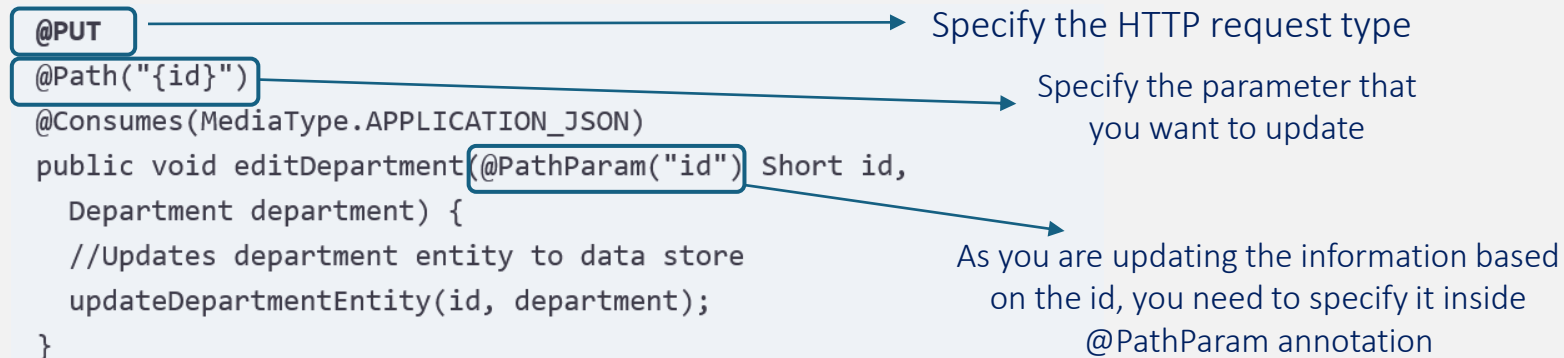
```
@Path("departments")
public class DepartmentService {
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Department> findAllDepartments() {
        //Find all departments from the data store
        List<Department> departments = findAllDepartmentsFromDB();
        return departments;
    }
    //Other methods removed for brevity
}
```

Specify the HTTP request type

Specify the media type

Annotations for processing HTTP request methods - PUT

- The [HTTP PUT method](#) is used for updating or creating the resource pointed by the URI.
- The [@javax.ws.rs.PUT](#) annotation designates a method of a resource class to respond to the HTTP PUT requests.
- When a request reaches a server, the framework intercepts the request and directs it to the appropriate method that matches the URI path and the HTTP method type.
- The following code snippet shows how you can use the [@PUT](#) annotation to designate the [editDepartment\(\)](#) method to respond to the HTTP PUT request.



```
@PUT
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
public void editDepartment(@PathParam("id") Short id,
    Department department) {
    //Updates department entity to data store
    updateDepartmentEntity(id, department);
}
```

Specify the HTTP request type

Specify the parameter that you want to update

As you are updating the information based on the id, you need to specify it inside @PathParam annotation

Annotations for processing HTTP request methods - POST

- The [HTTP POST method](#) posts data to the server. Typically, this method type is used for creating a resource.
- The [@javax.ws.rs.POST](#) annotation designates a method of a resource class to respond to the HTTP POST requests.
- The following code snippet shows how you can use the [@POST](#) annotation to designate the [createDepartment\(\)](#) method to respond to the HTTP POST request.

@POST —————> Specify the HTTP request type

```
public void createDepartment(Department department) {  
    //Create department entity in data store  
    createDepartmentEntity(department);  
}
```

Annotations for accessing request parameters - PathParam

```
javax.ws.rs.PathParam
```

```
@Path("departments")
```

```
public class DepartmentService {
```

```
    @DELETE
```

```
    @Path("{id}")
```

```
    public void removeDepartment(@PathParam("id") Short deptId) {
```

```
        removeDepartmentEntity(deptId);
```

```
    }
```

```
    //Other methods removed for brevity
```

```
}
```

If we want to match the parameter used inside the method with one specified in the Path, it must be included in the PathParam annotation

The REST API call to remove the department resource identified by id=10 looks like:

`DELETE /departments/10 HTTP/1.1.`

Annotations for accessing request parameters - PathParam

- We can also use multiple variables in a URI path template.
- For example, we can have the URI path template embedding the path variables to query a list of departments from a specific city and country, which may look like [/departments/{country}/{city}](#).
- The following code snippet illustrates the use of @PathParam to extract variable values from the preceding URI path template:

We can fetch the information of all departments according to a specific city

Inside the method, we need to pass both parameters using @PathParam

```
@Produces(MediaType.APPLICATION_JSON)
@Path("/{country}/{city}")
public List<Department> findAllDepartments(
    @PathParam("country")
    String countryCode, @PathParam("city") String cityCode) {
    //Find all departments from the data store for a country
    //and city
    List<Department> departments =
        findAllMatchingDepartmentEntities(countyCode,
            cityCode );
    return departments;
}
```

Annotations for accessing request parameters - QueryParam

- The `@javax.ws.rs.QueryParam` annotation injects the value(s) of a HTTP query parameter into a class field, a resource [class bean property \(the getter method for accessing the attribute\)](#), or a [method parameter](#).
- The following example illustrates the use of `@QueryParam` to extract the value of the desired query parameter present in the URI.
- This example extracts the value of the query parameter, name, from the request URI and copies the value into the deptName method parameter.
- The URI that accesses the IT department resource looks like [/departments?name=IT](#):

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Department>
    findAllDepartmentsByName(@QueryParam("name") String deptName) {
    List<Department> depts= findAllMatchingDepartmentEntities
        (deptName);
    return depts;
}
```

Here we specify a parameter to get the name of a specific department using `@QueryParam`

Annotations for accessing request parameters – MatrixParam

Provide a way to pass additional information within a URI, typically used to provide additional details about a resource.

```
@Path("/products/{category}/{productId}")
public class ProductResource {
    @GET
    public Response getProductDetails(
        @PathParam("category") String category,
        @PathParam("productId") int productId,
        @MatrixParam("color") String color,
        @MatrixParam("size") String size
    ) {
        // Implementation logic
    }
}
```

Example URI:

•/products/electronics/123;color=red;size=medium

Annotations for accessing request parameters - FormParam

- In JAX-RS, the `@FormParam` annotation is used to inject values submitted through an HTML form as part of an HTTP request.
- The `@javax.ws.rs.FormParam` annotation injects the matching HTML form parameters present in the request body into a class field.
- Note that the `@FormParam` annotation can only be used with HTTP POST requests that have the `application/x-www-form-urlencoded` media type.
- Consider the following HTML form that contains the data capture form for a department entity. This form allows the user to enter the department entity details

```
<!DOCTYPE html>
<html>
  <head>
    <title>Create Department</title>
  </head>
  <body>
    <form method="POST" action="/resources/departments">
      Department Id:
      <input type="text" name="departmentId">
      <br>
      Department Name:
      <input type="text" name="departmentName">
      <br>
      <input type="submit" value="Add Department" />
    </form>
  </body>
</html>
```


Annotations for accessing request parameters - FormParam

- Upon clicking on the submit button on the HTML form, the department details that you entered will be posted to the REST URI, /resources/departments.
- The following code snippet shows the use of the @FormParam annotation for extracting the HTML form fields and copying them to the resource class method parameter:

```
@Path("departments")
public class DepartmentService {

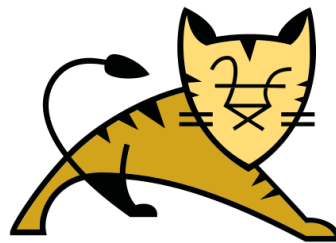
    @POST
    //Specifies content type as
    //"application/x-www-form-urlencoded"
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void createDepartment(@FormParam("departmentId") short
        departmentId,
        @FormParam("departmentName") String departmentName) {
        createDepartmentEntity(departmentId, departmentName);
    }
}
```



Tools for JAX-RS

Apache Tomcat

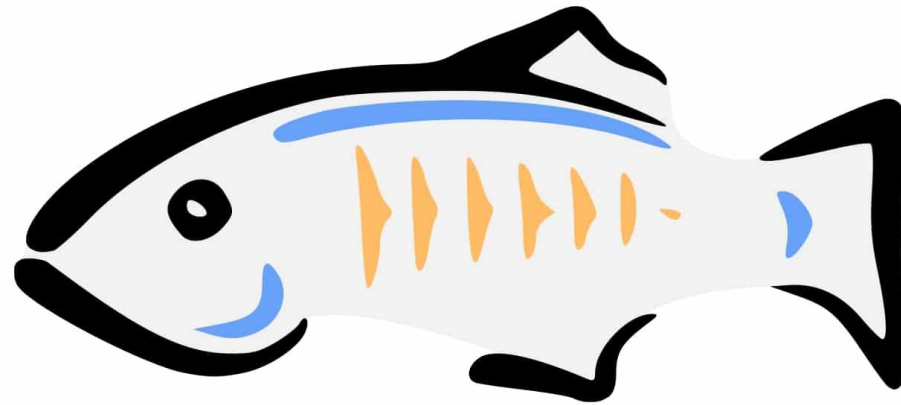
- [Apache Tomcat](#) is an open-source web server and servlet container developed by the Apache Software Foundation.
- Tomcat is lightweight and easy to use, making it a popular choice for developing and deploying Java-based web applications.



Apache Tomcat

Glassfish

- GlassFish is an open-source application server project that provides a robust and scalable platform for developing, deploying, and managing Java EE (Enterprise Edition) and Jakarta EE (the successor to Java EE) applications.
- It is developed by the Eclipse Foundation and has a long history, initially being a project sponsored by Sun Microsystems and later Oracle.



Wildfly server

- WildFly is a free and open-source application server written in Java, originally developed by Red Hat.
- It provides a runtime environment for Java-based applications and supports a wide range of Java EE (Enterprise Edition) technologies, including web services, messaging, and security.
- WildFly is known for its speed, flexibility, and high-performance capabilities, and is often used in enterprise-level applications that require robust and scalable infrastructure.



Which one is better?



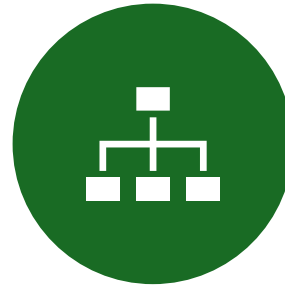
Lightweight application
development



Scalability



Integration



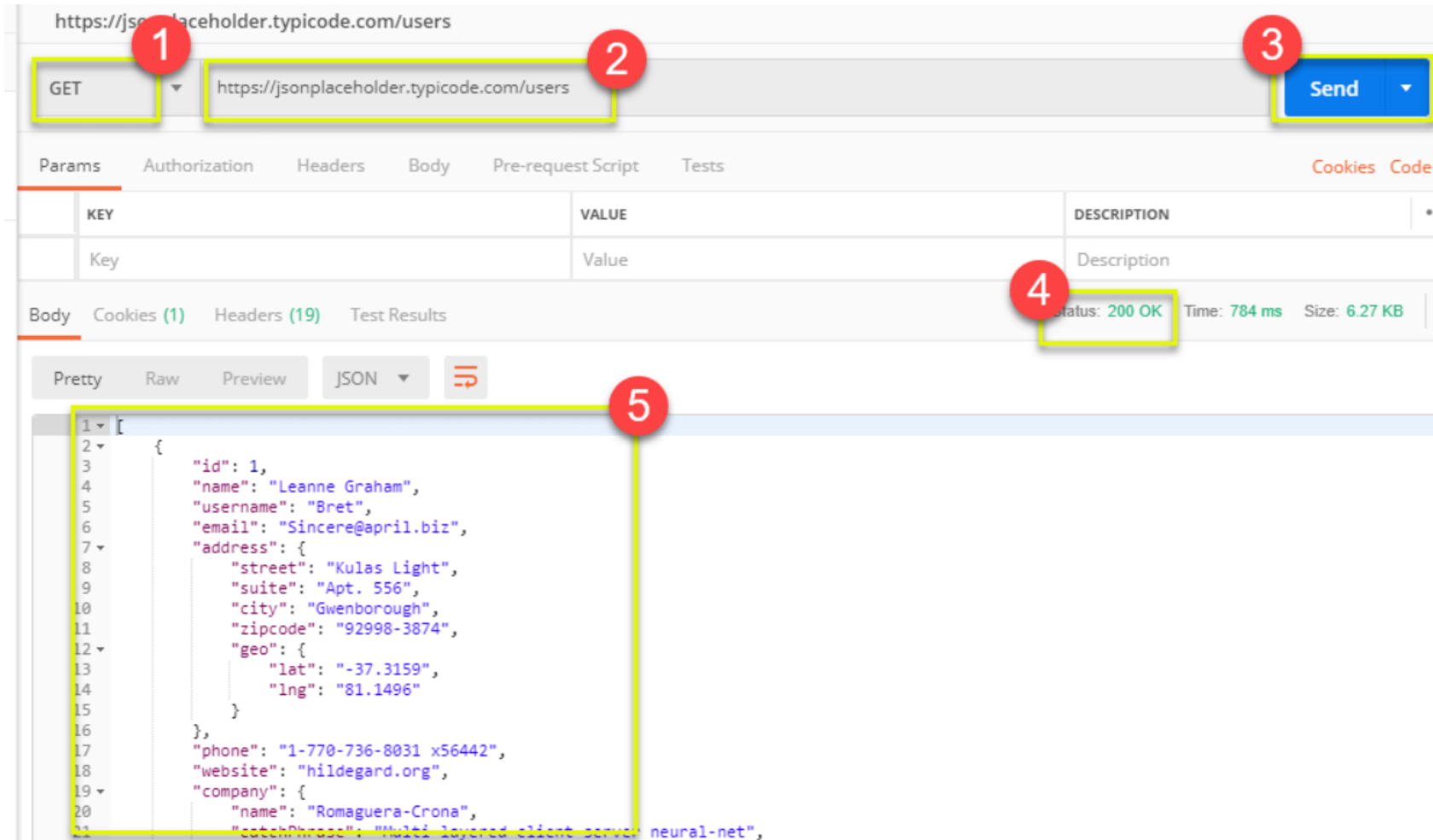
Administration

Postman

- Postman is a popular collaboration platform for building, testing, and documenting Application Programming Interfaces (APIs).
- It provides a user-friendly interface for making HTTP requests to APIs
- Using Postman, developers can create requests with different HTTP methods (such as GET, POST, PUT, DELETE, etc.), add headers and parameters, and send data in different formats (such as JSON, XML, form data, etc.).



Postman - example





NEW TO POSTMAN ??

WHAT IS POSTMAN ??



HOW TO USE POSTMAN ??



HOW TO TEST API ??

