

### Tutorial Week 07: RESTful web services with JAX-RS

#### INTRODUCTION

Java API for RESTful Web Services (JAX-RS) is a set of APIs and specifications that simplify the development of RESTful web services in Java. REST (Representational State Transfer) is an architectural style for designing networked applications, and JAX-RS provides a Java-based approach for implementing RESTful services.

#### 1. Purpose of JAX-RS:

- JAX-RS simplifies the creation of web services that follow REST principles.
- It provides a set of annotations and APIs for building scalable and flexible web services in Java.

#### 2. RESTful Web Services:

- RESTful services are designed around the concept of resources, which can be identified and manipulated using standard HTTP methods (GET, POST, PUT, DELETE).
- These services emphasize statelessness, scalability, and a uniform interface.

#### 3. Java EE and Jakarta EE:

- JAX-RS is part of the Java EE (Enterprise Edition) and Jakarta EE (successor to Java EE) specifications.
- Java EE and Jakarta EE are comprehensive enterprise platforms that provide a range of technologies for building scalable and distributed applications.

#### 4. Modularity and Extensibility:

- JAX-RS allows developers to create modular and extensible applications by leveraging features like resource classes, annotations, and providers.
- It supports the creation of both client and server components for interacting with RESTful services.

#### 5. Annotation-Based Programming:

- One of the strengths of JAX-RS is its use of annotations for defining various aspects of a web service, such as resource paths, HTTP methods, and parameter extraction.

## REQUIREMENTS

- NetBeans IDE
- Apache Tomcat 9.x.x
- Postman Desktop API

## SETTING UP THE DEVELOPMENT ENVIRONMENT

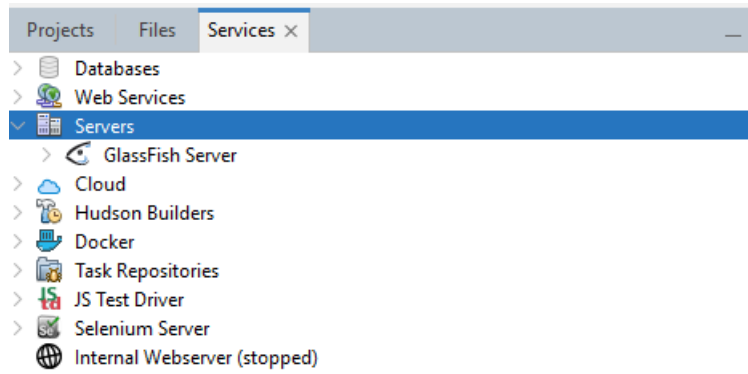
To start implementing any JAX-RS application, we need to add a web application server. For all tutorials, we will use **Apache Tomcat**.

Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation. It is designed to execute Java Servlets and render Java Server Pages (JSP), providing a robust environment for deploying and running Java web applications. Here are key points to understand about Apache Tomcat:

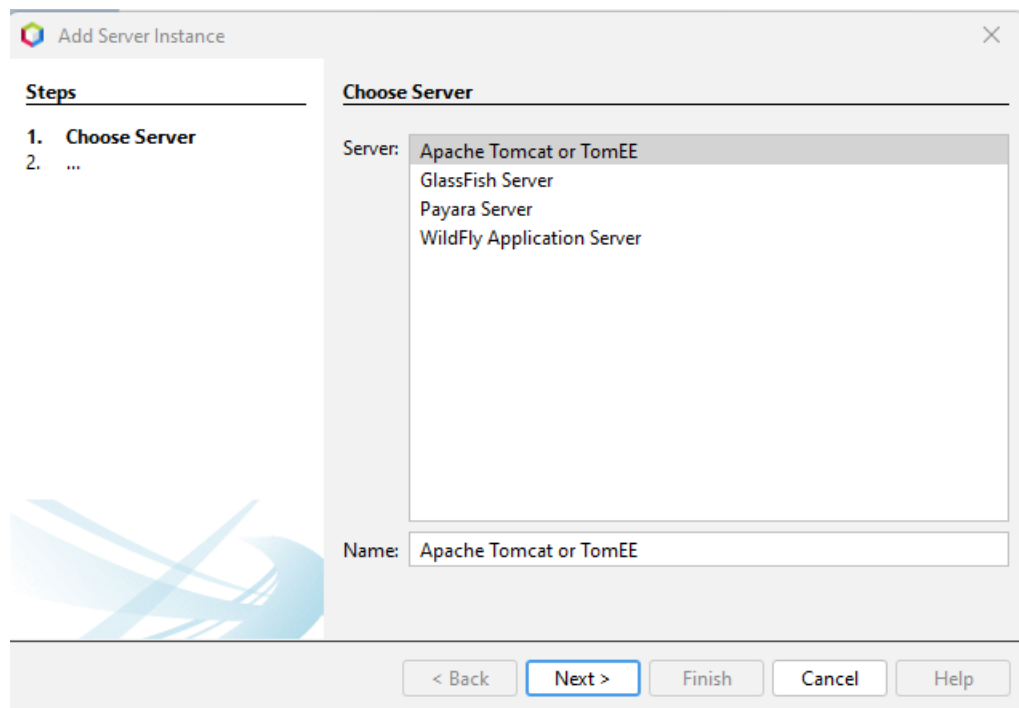
- **Servlet Container:**
  - Apache Tomcat serves as a servlet container, implementing the Java Servlet.
  - A servlet container is a part of web servers responsible for managing the execution of servlets, which are Java classes that handle HTTP requests and responses.
- **Open Source:**
  - Apache Tomcat is open-source software, released under the Apache License.
  - Being open-source allows developers to access and modify the source code, contributing to its improvement and customization.
- **Java EE Compatibility:**
  - While Apache Tomcat is not a full-fledged Java EE (Enterprise Edition) application server, it is often used for deploying Java web applications that adhere to the Servlet.
  - It is lightweight compared to some other Java EE application servers, making it suitable for simpler applications and development environments.
- **HTTP Server Capabilities:**
  - In addition to its role as a servlet container, Tomcat can also function as a standalone web server capable of handling HTTP requests and responses.
  - It supports the basic HTTP features and can be used as a lightweight alternatives like GlassFish and WildFly.

## Set up Apache Tomcat server

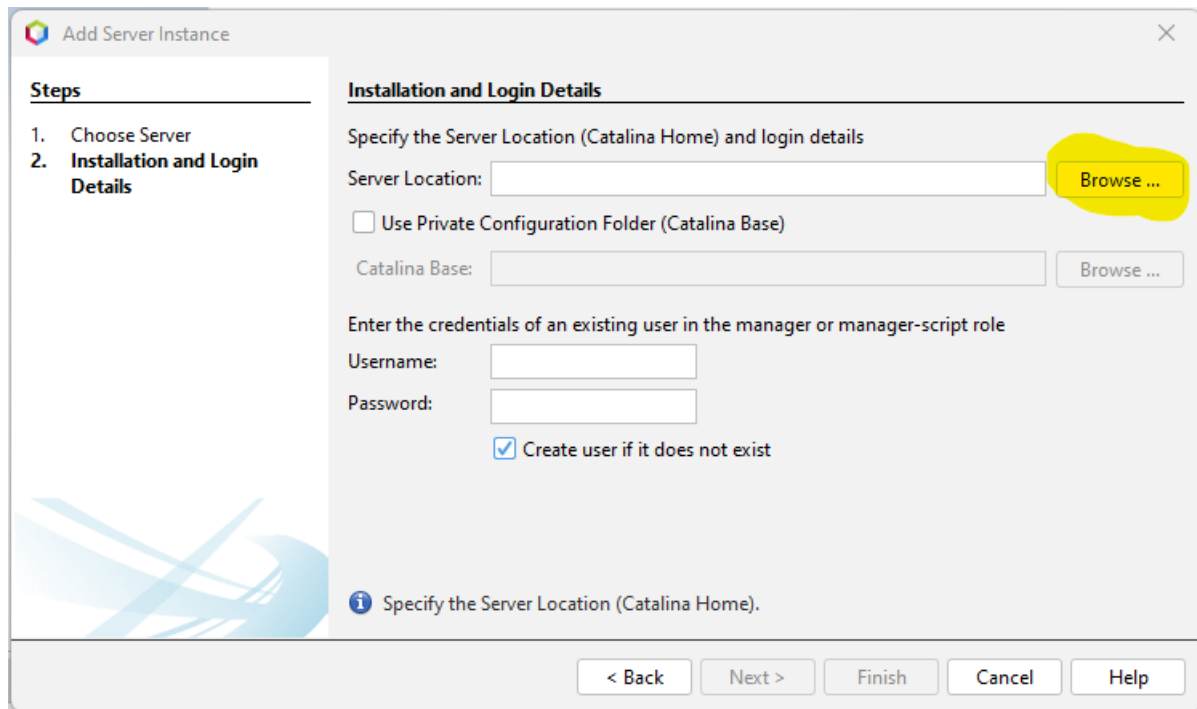
1. Download the Apache Tomcat server provided as a ZIP file under Week07 in the Blackboard.
2. Then extract it in a specific location
3. In the Services tab, right-click on the Server and then click on Add Server



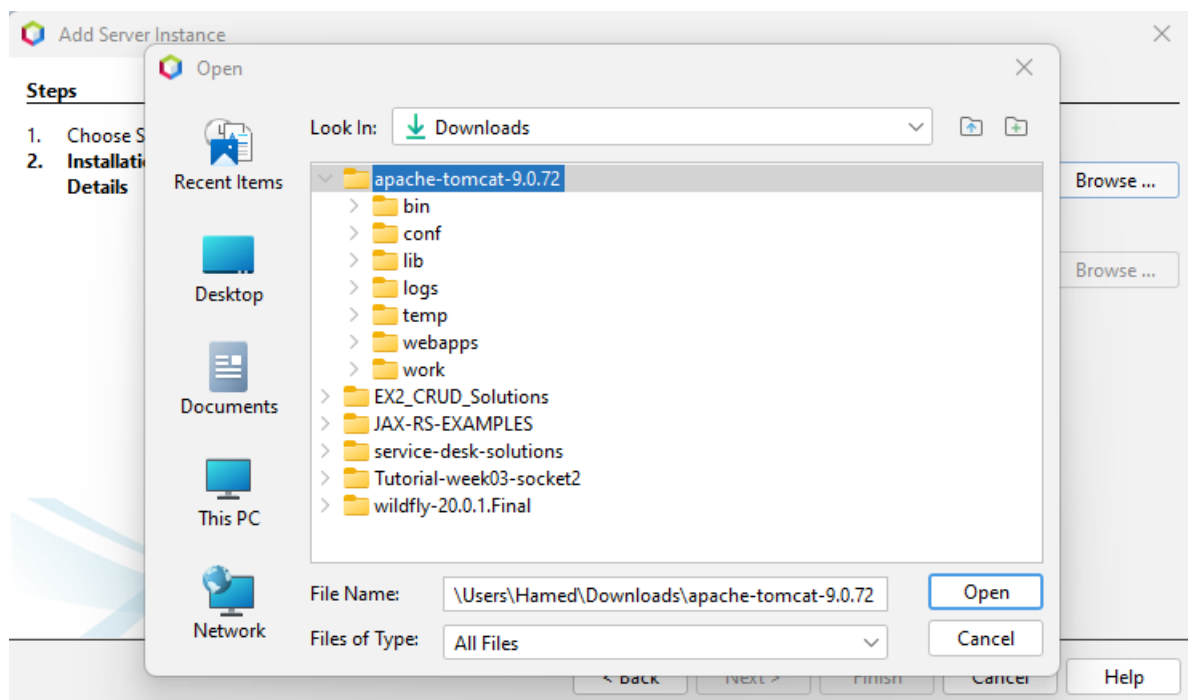
4. In the **Add Server Instance** window, select **Apache Tomcat or TomEE** and then click on **next**.



- Then, click on **Browse** and locate the Tomcat Server folder that you already extracted it.



- After locating the extracted file, click on **apache-tomcat-9.0.72** and then click on **Open**.



7. After locating the file, provide username and password. You can use anything as you wish. Here I'm using my name for both fields. Finally, click on finish.

The screenshot shows the 'Add Server Instance' dialog box with the 'Installation and Login Details' tab selected. The 'Steps' pane on the left shows '1. Choose Server' and '2. Installation and Login Details'. The main area contains the following fields and options:

- Server Location:** C:\Users\Hamed\Downloads\apache-tomcat-9.0.72 (with a 'Browse ...' button)
- ☐ Use Private Configuration Folder (Catalina Base)
- Catalina Base:** (empty field with a 'Browse ...' button)
- Enter the credentials of an existing user in the manager or manager-script role**
- Username:** hamed
- Password:** ..... (masked)
- ☒ Create user if it does not exist

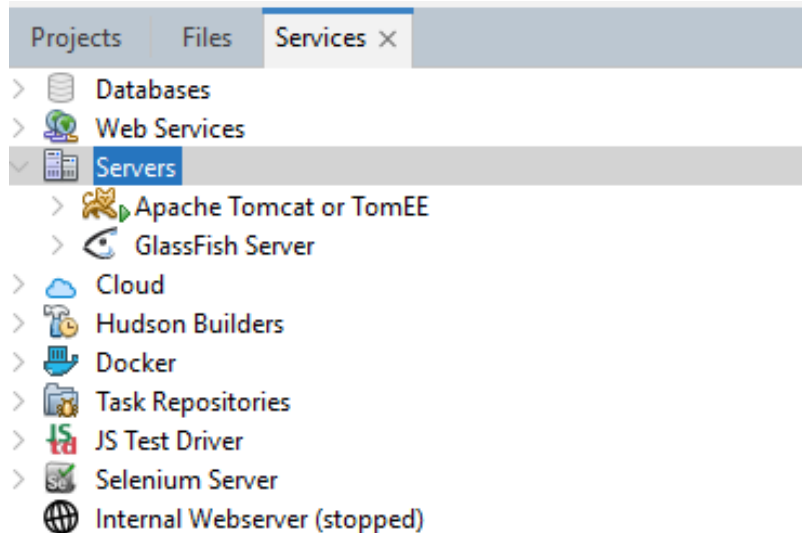
At the bottom, there are buttons: '< Back', 'Next >', 'Finish' (highlighted), 'Cancel', and 'Help'.

8. After clicking on Finish, the Apache Tomcat asks you to enter the credentials that you provided in the previous step. Then, click on ok. Please note, sometimes if you have existing passwords for the Apache Tomcat, it will ask to remove them. If you are asked to do so, please click on ok.

The first screenshot shows an 'Authentication Required' dialog box for the 'Tomcat Manager Application'. It contains fields for 'User Name' (hamed) and 'Password' (masked with dots), with 'OK' and 'Cancel' buttons at the bottom.

The second screenshot shows a 'Delete Stored Passwords' dialog box. It contains a question mark icon and the text: 'Your previously stored passwords cannot be decrypted. Do you wish to forget all existing passwords and start with a fresh keyring?'. It has 'OK' and 'Cancel' buttons at the bottom.

9. If you check the **Services** tab, you should see a small green triangle which means that the Server is up and running. There you go!. You are ready to start with implementation.



---

#### USING APACHE TOMCAT IN MAC

If you are using Apache Tomcat in Mac systems, please note that the file named Catalina.sh must get permission to be executed. For this you need to open the terminal and locate the folder for Apache tomcat where you have extracted. For example, if you have extracted it to Downloads folder you need to type the following commands in terminal one by one:

```
cd Downloads
```

```
cd apache-tomcat-9.0.100
```

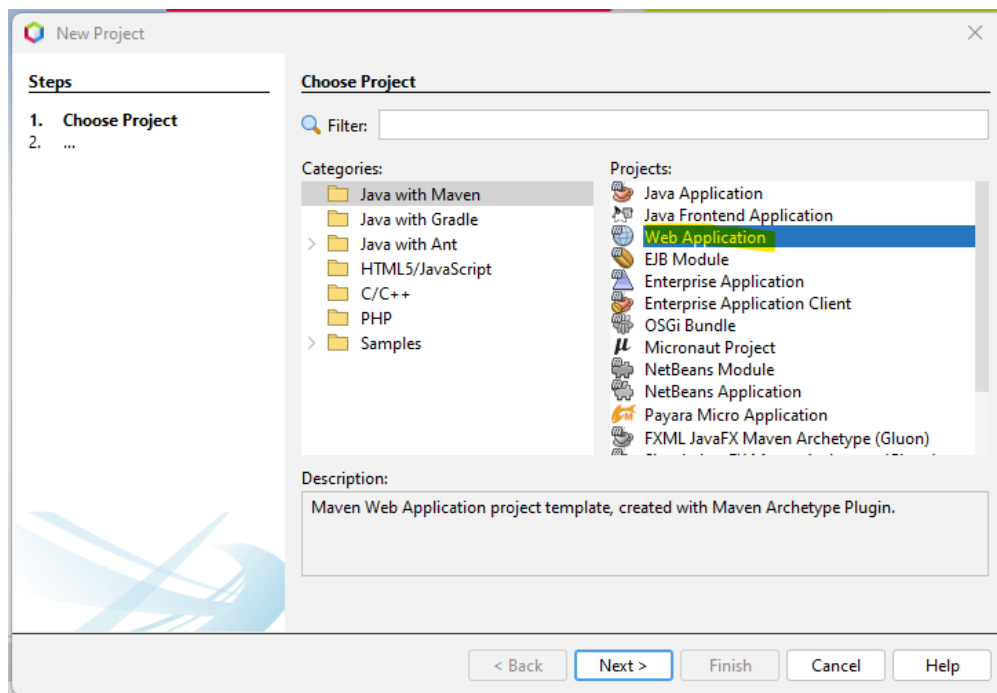
```
cd bin
```

```
chmod +x catalina.sh startup.sh shutdown.sh
```

Then, please start the server. If it does not work, please remove the server from NetBeans and add it again and start it.

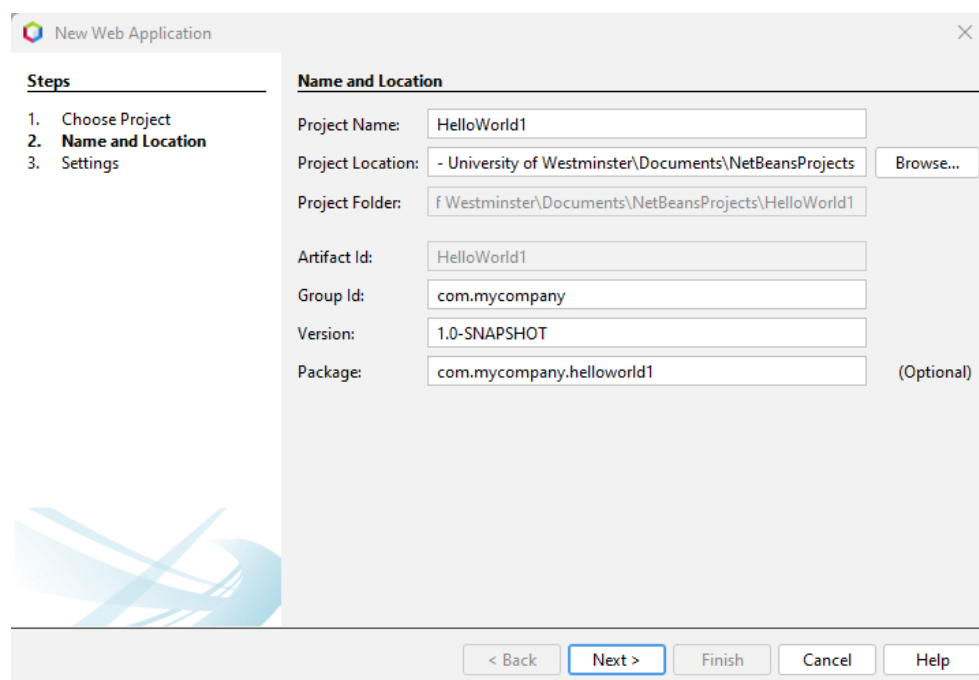
#### CREATE A NEW PROJECT

Open NetBeans and create a new project. Select Java with Maven -> Web Application.



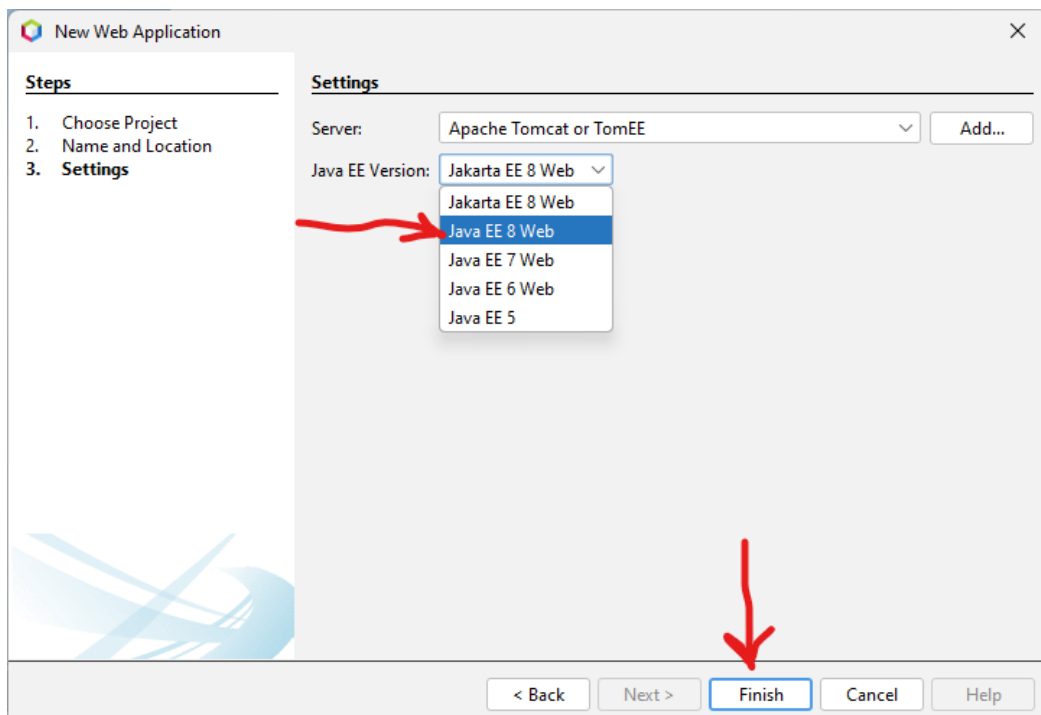
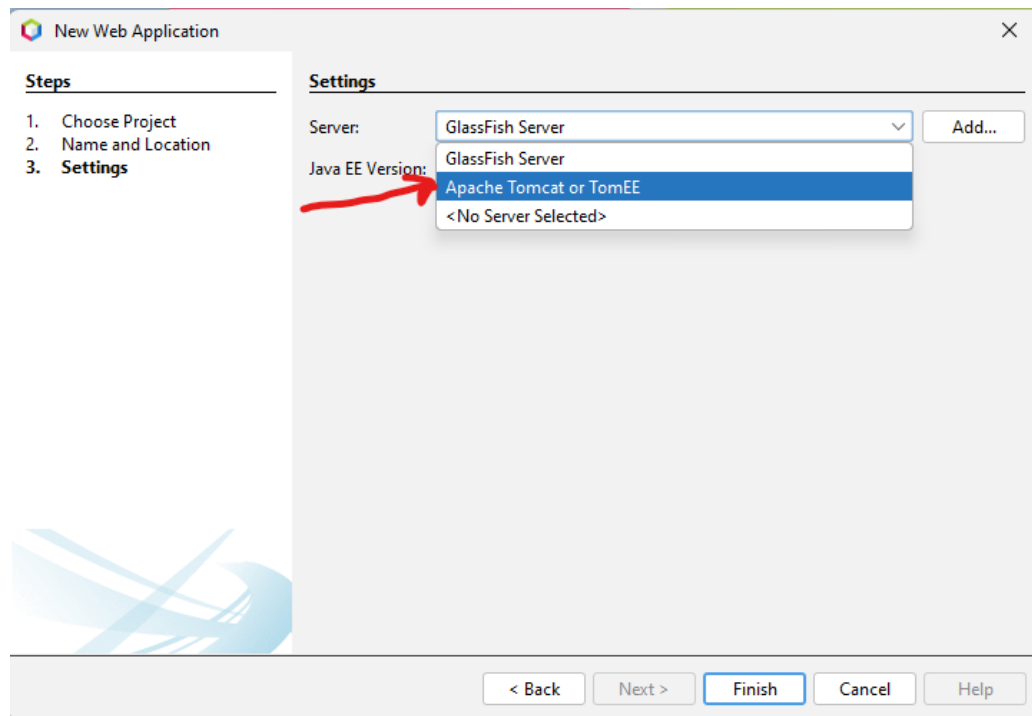
## SET UP THE PROJECT

Name your project as “HelloWorld1”



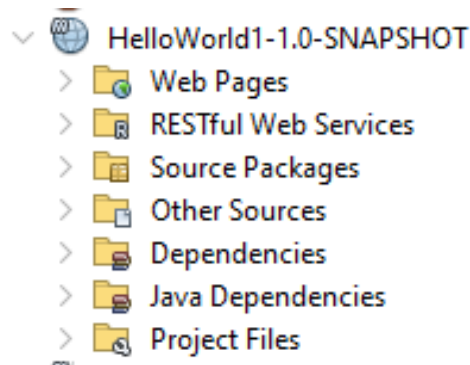
## SELECT A SERVER

After clicking on the Next, you will be asked to select a Server to deploy your application. To do so, please select Apache Tomcat or TomEE and then select the Java EE 8 Web. **Please note, if you have not installed Apache Tomcat, it will not be available in the menu.**





2. **Project architecture:** After clicking on Finish, if you check the Projects tab, the structure of the application must look like as follows:



## ADDING DEPENDENCIES

Dependencies in a JAX-RS (Java API for RESTful Web Services) application must include external libraries or modules that provide the required functionality for building, deploying, and running the application. These dependencies are typically packaged as JAR (Java Archive) files and are added to the project to bring in additional features, libraries, or frameworks. Here are several reasons why adding dependencies to a JAX-RS application is essential:

### 1. Modularization and Code Reusability:

- Dependencies allow developers to modularize their applications. Instead of reinventing the wheel for common functionalities, developers can include existing libraries developed, tested, and proven to work well.
- Reusing well-established libraries promotes code reusability and saves development time.

### 2. Framework Support:

- JAX-RS itself is a set of APIs, but it requires an implementation to be used in an actual application. Popular implementations include Jersey, RESTEasy, and Apache CXF. Adding the appropriate implementation as a dependency provides the runtime environment for your JAX-RS application.

### 3. Simplify Development:

- Dependencies often simplify the development process by providing high-level abstractions and utilities. For example, adding a JSON processing library as a

dependency simplifies the task of marshalling and unmarshalling JSON data in your JAX-RS application.

#### 4. Connectivity and Integration:

- JAX-RS applications often need to interact with databases, messaging systems, or other external services. Dependencies for JDBC drivers, JMS libraries, or other connectors enable seamless integration with these systems.

#### 5. Security Features:

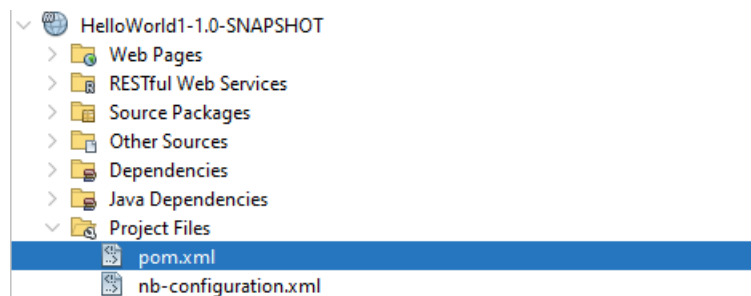
- Security is a critical aspect of web applications. Dependencies related to security, such as authentication and authorization libraries, help in implementing secure access control mechanisms in your JAX-RS application.

#### 6. Testing Frameworks:

- Dependencies on testing frameworks like JUnit or TestNG are crucial for writing and executing unit tests for your JAX-RS resources and components. Testing ensures the reliability and correctness of your application.

### STEP-BY-STEP GUIDE ON HOW TO ADD DEPENDENCIES TO A JAX-RS APPLICATION IN NETBEANS 18

1. **Open the Project's POM.xml file:** In the Projects window, expand your project and locate the **pom.xml** file. This is the Project Object Model (POM) file for Maven projects, which is used to manage project dependencies.



2. **Edit the POM.xml file:** Double-click the pom.xml file to open it. You'll see an XML structure. You need to add your dependencies inside the <dependencies> tag.
3. **Add the JAX-RS dependencies:** For your JAX-RS application, please edit the file using the following dependencies:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>PACKAGE_NAME</groupId>
  <artifactId>PROJECT_NAME</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>org.glassfish.jersey.containers</groupId>
      <artifactId>jersey-container-servlet</artifactId>
      <version>2.32</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.inject</groupId>
      <artifactId>jersey-hk2</artifactId>
      <version>2.32</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>8</source>
          <target>8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <name>HelloWorld1-1.0-SNAPSHOT</name>
</project>
```

---

## MAVEN DEPENDENCIES

### JERSEY CONTAINER SERVLET:

- **Group ID:** org.glassfish.jersey.containers

- **Artifact ID:** jersey-container-servlet
- **Version:** 2.32
- **Purpose:** This dependency includes the Jersey implementation of the servlet container. It provides classes and components necessary for integrating Jersey with a servlet-based web application.

#### JERSEY HK2 (HUNDRED KILOBYTES KERNEL):

- **Group ID:** org.glassfish.jersey.inject
- **Artifact ID:** jersey-hk2
- **Version:** 2.32
- **Purpose:** HK2 is the dependency injection framework used by Jersey. This dependency includes the necessary components for integrating HK2 with Jersey, enabling dependency injection within your JAX-RS resources and components.

---

### MAVEN BUILD PLUGINS

#### MAVEN COMPILER PLUGIN:

- **Group ID:** org.apache.maven.plugins
- **Artifact ID:** maven-compiler-plugin
- **Version:** 3.8.1
- **Purpose:** The Maven Compiler Plugin is used to configure and customize the behavior of the Java compiler during the build process. In this case, it sets the Java version to be used by the compiler to version 8, both for source code and target bytecode.

---

#### MAVEN WAR PLUGIN:

- **Group ID:** org.apache.maven.plugins
- **Artifact ID:** maven-war-plugin
- **Version:** 3.2.2
- **Purpose:** The Maven WAR Plugin is used to build and package the project as a Web Application Archive (WAR) file. In this configuration, it disables the requirement for a **web.xml** file:

(`<failOnMissingWebXml>false</failOnMissingWebXml>`), which is common in modern servlet containers and JAX-RS applications that use annotated classes for configuration.

## FINDING MAVEN DEPENDENCIES

In order to find maven dependencies you need to visit [maven repository web site](#). Once you locate the website, you need to enter any dependencies that you need to include inside pom.xml file.

For example, to find jersey servlet, we need to search for this in the repository as follows:

MVN REPOSITORY

jersey container ~~servlet~~

Search

Repository

Central6.6k

Spring Lib M1.6k

Sonatype1.5k

Spring Plugins1.5k


JBoss Releases609

JCenter312

IBiblio262

Found 8438 results

Sort: **relevance** | popular | newest



1. Jersey Container Servlet

org.glassfish.jersey.containers » jersey-container-servlet

Jersey core Servlet 3.x implementation

Last Release on Dec 12, 2023

736 usages

CC0BSDApache

Once you locate it, click on and then select version 2.32 which is compatible with the Apache Tomcat version 9.

2.37.x	<a href="#">2.37</a>	Central	29	Sep 06, 2022
2.36.x	<a href="#">2.36</a>	Central	42	Jun 14, 2022
2.35.x	<a href="#">2.35</a>	Central	71	Sep 07, 2021
2.34.x	<a href="#">2.34</a>	Central	70	Apr 20, 2021
2.33.x	<a href="#">2.33</a>	Central	38	Dec 18, 2020
2.32.x	<a href="#">2.32</a>	Central	61	Sep 25, 2020
2.31.x	<a href="#">2.31</a>	Central	62	May 22, 2020

After choosing the specific version for servlet container, then you can copy the maven content shown as follows and paste it inside the pom.xml file.

MavenGradleGradle (Short)Gradle (Kotlin)SBTLvyGrapeLeiningenBuildr

```
<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.containers/jersey-container-servlet -->
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.32</version>
</dependency>
```

## SAVE POM.XML FILE

After editing and adding the dependencies, please save the **pom.xml** file.

## SETTING UP RESOURCE CLASS

1. Open Source Packages and under **helloworld1** package, create a class called **HelloWorldResource**
2. Remove **JAXRSConfiguration.java**

### STEP 1: IMPORT NECESSARY LIBRARIES

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

**Get:** It is used to annotate a Java method to indicate that it will handle HTTP GET requests.

**Path:** It is Used to annotate a Java class or method to indicate the base URI path for which the resource or resource method will handle requests.

**Produces:** It is Used to specify the media types (MIME types) that the resource method can produce as output.

**MediaType:** It is class that defines standard media types (MIME types). It is often used in conjunction with @Produces to specify the expected response format.

## STEP 2: SPECIFYING RESOURCE AND HTTP METHOD

```
@Path("/hello")

public class HelloWorldResource {

    @GET

    @Produces(MediaType.TEXT_PLAIN)

    public String getHello() {

        return "Hello, World!";

    }

}
```

---

### CODES BREAKDOWN:

#### 1. **@Path("/hello"):**

- This annotation is applied to the class and specifies the base URI path for which this resource class will handle requests. In this case, it indicates that this resource class will handle requests for the path `"/hello"`.

#### 2. **Resource Class (HelloWorldResource):**

- This is a simple Java class that acts as a JAX-RS resource. It is annotated with **@Path("/hello")**, and all methods within this class will handle requests under the base path `"/hello"`.

#### 3. **@GET:**

- This annotation is applied to the **getHello** method, indicating that this method will handle HTTP GET requests.

#### 4. **@Produces(MediaType.TEXT\_PLAIN):**

- This annotation is used to specify the media type (MIME type) that the method can produce as output. In this case, it indicates that the **getHello** method produces plain text (**text/plain**) as its response.

#### 5. **public String getHello() { ... }:**

- This is the method that will be invoked when an HTTP GET request is made to the `"/hello"` path. It returns a simple string, `"Hello, World!"`, as the response.

## SETTING UP APPLICATION PATH CLASS

Create a new class called **MyApplication** under the same package and add the following packages

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
import java.util.HashSet;
import java.util.Set;
```

---

### CODES BREAKDOWN:

#### JAVAX.WS.RS.APPLICATIONPATH

- This package contains the `ApplicationPath` annotation. This is crucial for defining the base URI path for all your REST resources within the application.
- **Example:** If you annotate a class with `@ApplicationPath("/api")`, all your REST resources' URIs will begin with `/api`.

---

#### JAVAX.WS.RS.CORE.APPLICATION

- This package contains the core `Application` class. This class acts as the central configuration point or "blueprint" for your JAX-RS application. By extending it, you can:
  - Register your REST resources.
  - Define providers for custom data type handling, exception mapping, etc.

---

#### JAVA.UTIL.HASHSET AND JAVA.UTIL.SET

- These come from the Java Collections Framework.
  - `Set` is an interface that represents a collection of unique elements.
  - `HashSet` provides a concrete implementation of `Set` using a hash table for efficient storage and retrieval.

Then, please complete the rest by adding the following codes:

```
@ApplicationPath("rest")
public class MyApplication extends Application {
    @Override
    public Set<Class<?>> getClasses() {
```



```

        Set<Class<?>> classes = new HashSet<>();

        classes.add>HelloWorldResource.class);

        return classes;
    }
}

```

---

## CODES BREAKDOWN:

### 1. @ApplicationPath("rest")

- This annotation on the MyApplication class defines the root path for all your RESTful resources within the application. With this, all your resource URIs would be prefixed with /rest.

### 2. public class MyApplication extends Application { ... }

- MyApplication is the core configuration class of your REST application. It extends the javax.ws.rs.core.Application class, providing a place to manage the registration of resources and other configuration aspects.

### 3. @Override public Set<Class<?>> getClasses() { ... }

- The getClasses() method is a crucial part of configuring a JAX-RS application. By overriding it, you specify the REST resource classes the JAX-RS runtime should manage.
- **Inside the method:**
  - Set<Class<?>> classes = new HashSet<>(); : Creates a HashSet to store references to your resource classes. Using a HashSet ensures uniqueness and efficient lookups.
  - classes.add>HelloWorldResource.class); : Adds the HelloWorldResource class to the set. This implies that HelloWorldResource contains your REST endpoint definitions.
  - return classes;; Returns the set of registered resource classes.

## APPLICATION CONFIGURATION CLASS

We need configuration because of the following reasons:

- **DEFINING THE STRUCTURE OF YOUR REST API:**

JAX-RS, at its core, provides the building blocks for creating RESTful services. The configuration acts as the architectural blueprint for your API. It's where you tell the JAX-RS runtime:

- **Base path:** The root URL segment that your REST endpoints will share (e.g., /api, /rest).
- **REST resources:** Which classes contain your actual RESTful endpoint logic. These classes handle incoming requests and produce responses.
- **Providers:** Components that can handle custom data conversions, exception mapping, security, and more.

- **DISCOVERY AND MANAGEMENT**

JAX-RS implementations need a way to understand how to turn your Java classes into a functional REST API. The configuration provides a structured way for the runtime to:

- Scan for classes marked with relevant JAX-RS annotations (@Path, @GET, etc.).
- Understand which URL patterns map to which methods.
- Manage the lifecycle of your resources and providers.

- **CUSTOMIZATION**

The configuration layer lets you tailor the behavior of your REST API. You can:

- Register filters or interceptors to manipulate requests or responses.
- Provide custom message converters for handling non-standard data formats.
- Configure security mechanisms.

**Now, let's implement the class by taking the following steps:**

**STEP 1: FIRST OF ALL ADD THE FOLLOWING LIBRARY:**

```
import org.glassfish.jersey.server.ResourceConfig;
```

- **org.glassfish.jersey.server.ResourceConfig:** This class is a core part of the Jersey framework, which is a popular implementation of the JAX-RS (Java API for RESTful Web Services) specification. Here's what it does:
  - **Central Configuration Point:** The ResourceConfig class acts as the primary configuration mechanism for your Jersey-based REST applications. You extend this class (or use it directly) to define the essential components of your API.

- **Key Functionality:** Within a ResourceConfig class, you can:
  - Register your REST resource classes (the classes containing @Path annotations and HTTP method annotations).
  - Register providers (for custom message body handling, exception mapping, etc.).
  - Configure features (security, filters, etc.)
  - Set properties to tune the behavior of the Jersey runtime.

## STEP 2: IMPLEMENT CONFIGURATION LOGIC:

```
public class MyApplicationConfig extends ResourceConfig {  
    public MyApplicationConfig() {  
        register(HelloWorldResource.class);  
    }  
}
```

---

## CODES BREAKDOWN:

### 1. public class MyApplicationConfig extends ResourceConfig:

This defines a class named MyApplicationConfig. The key point is that it extends the ResourceConfig class provided by the Jersey framework. ResourceConfig acts as a streamlined configuration mechanism specifically designed for resource registration.

### 2. public MyApplicationConfig() { ... }

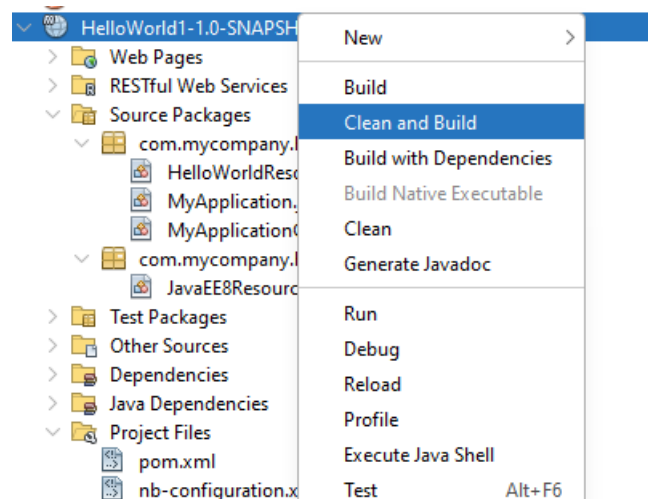
This is the constructor of your MyApplicationConfig class. It's where core initialization logic happens.

### 3. register(HelloWorldResource.class);

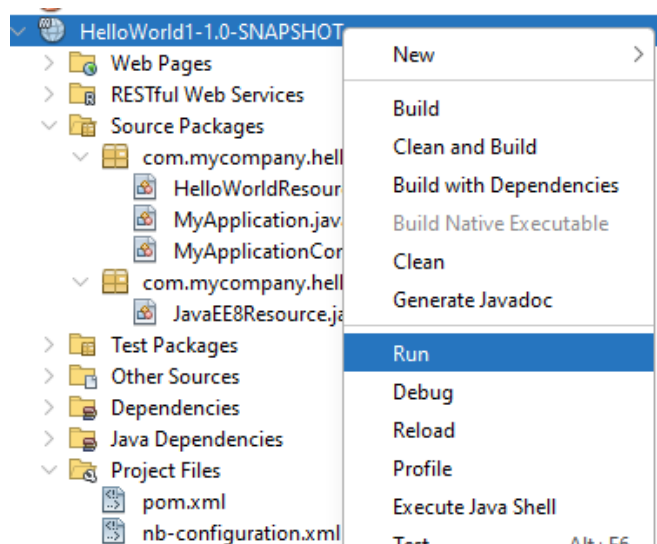
This is the critical line inside the constructor. The register() method instructs the Jersey framework to directly manage the HelloWorldResource class as a RESTful resource. This means that the JAX-RS runtime will recognize this class and its methods that likely have JAX-RS annotations (@Path, @GET, etc.).

## RUN THE APPLICATION

We are now ready to test the first RESTful service developed by JAX-RS. To do this, first of all please do right-click on the project and click on **Clean and Build**. This allows us to make sure that all dependencies are installed properly.



Once we ensure that the build process is successful, then you can do right-click on the project and this time click on **Run**.

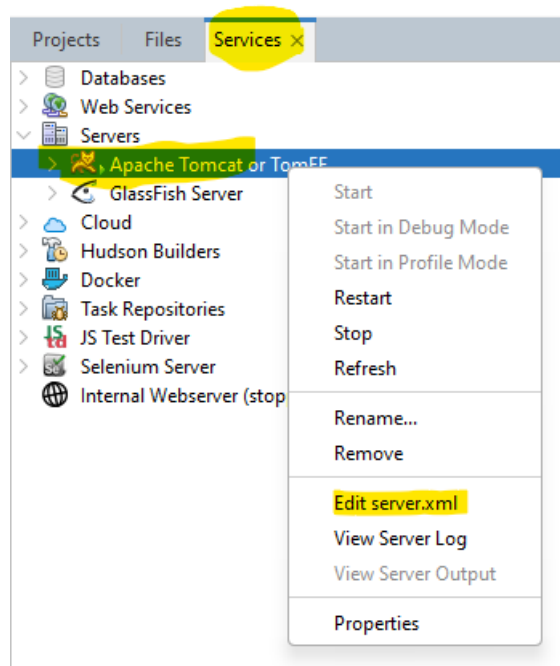


## TEST THE APPLICATION

Now, you can test you application using the following URI:

**localhost:8080/HelloWorld1/rest/hello**

Please note that, in this URI the port number is 8080 that must be the same as one specified in the Apache Tomcat configuration. To check the connector port number in the Apache Tomcat, please do right-click on the Apache Tomcat in the Services tab and then select **Edit server.xml**

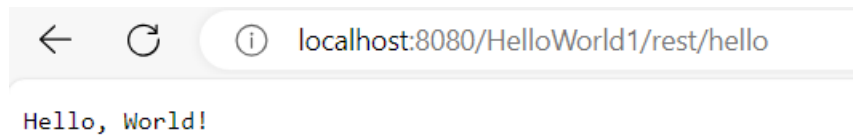


Once you locate the following lines inside the **server.xml**, you can check if the port number in the URI is the same as the number in the tomcat Connector port.

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
```

Finally, if everything is OK, you expect to see the following result shown in the web browser.

Please not that once you run the application a default browser will be automatically launched after a couple of seconds.



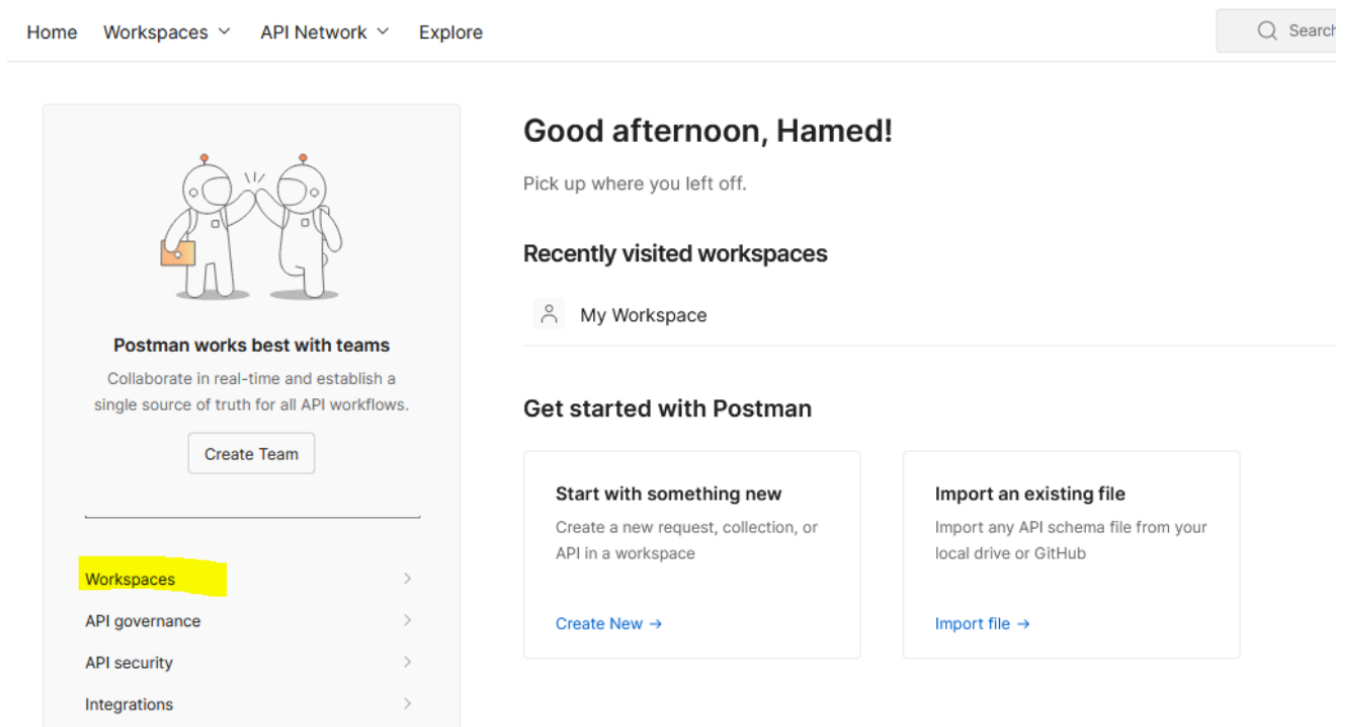
## TESTING THE SERVICE USING POSTMAN

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster. Using the Postman you can call HTTP methods in your RESTful service in an interactive interface. You can also download the Postman as use it as a desktop app.

- To use Postman, you need to sign up to it using the following link:

<https://identity.getpostman.com/signup> 2.

- After signing up. Use your credentials to login to the service. 3. Then you need to create a workspace by clicking on the workspaces



- Then you can create a workspace by clicking on Create Workspace. Then specify a name for that like test-api and set it up to Personal

## Create workspace

Name

test-api

Summary

Add a brief summary about this workspace.

Visibility

Determines who can access this workspace.



Personal

Only you can access



Private

Only invited team members can access



Team

All team members can access



Partner

NEW

Only invited partners and team members can access



Public

Everyone can view

Create Workspace

Cancel

- Click on “+” specify by yellow colour.

Overview



test-api

Add a brief summary about this workspace

Add a description to explain all about this workspace

Activity ↻

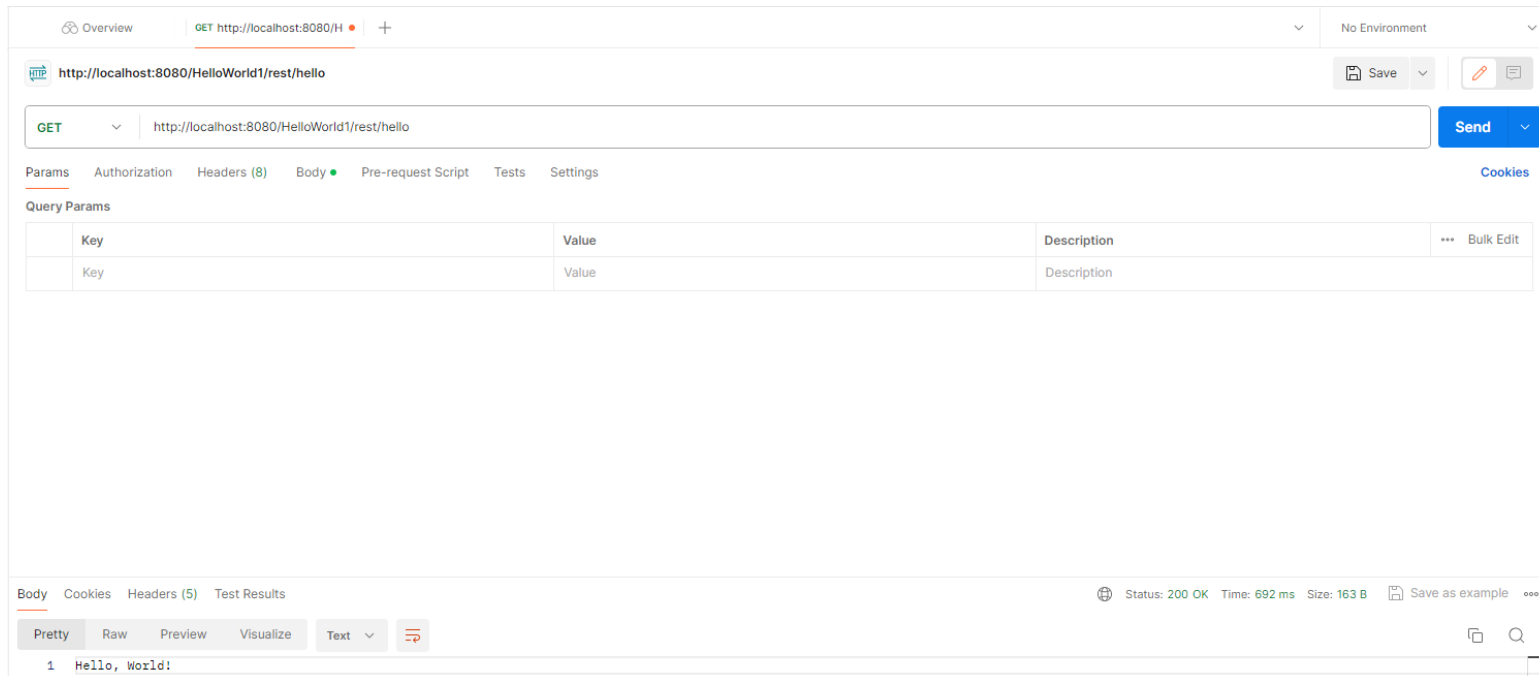
Filter by Elements ▾ People ▾

Today



Hamed created this personal workspace  
just now

- Then, you are ready to go. The only thing you need to do is to include the full URI in the box and click on **Send** to see the effects. As we have only implemented GET method, you can only send GET request.



Please note that if you want to use the Postman to test your API in the lab systems, you need to either download Windows or Mac-based platforms and use a **lightweight version** after running the exe file. [Download Postman | Get Started for Free](#)

## Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

### The Postman app

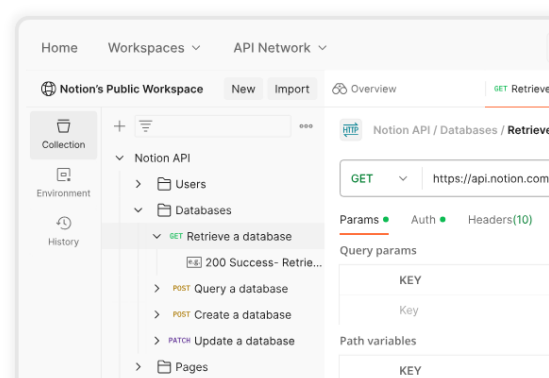
Download the app to get started with the Postman API Platform.

Windows 64-bit

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

[Release Notes](#) →

Not your OS? Download for Mac ([Intel Chip](#), [Apple Chip](#)) or Linux ([x64](#), [arm64](#))





## EXERCISE 1

In the previous example, the application returns an output as plain text. In this exercise, you will make some changes to produce JSON as a return type. To do this exercise, please do the following steps:

### COPY THE PREVIOUS PROJECT

Please copy the previous project by right-clicking on the project and selecting copy. Usually, a copy of the project will be created with the same name. If this happens, please right-click again on the project and select rename and then please provide a new name for that, like **HelloWorld1**.

### ADDING DEPENDENCY

- Locate the dependency for **jersey-media-json-jackson** version 2.32 in MVN repository and add it to pom.xml file.

### CLASS MODIFICATION

Please modify the HelloWorldResource class considering the following criteria:

#### 1. Define a JSON Response Class:

- Create a nested static class within **HelloWorldResource** named **HelloResponse**.
- Add a private String field named **message**.
- Create a constructor that takes a String parameter and initializes the **message** field.
- Implement getter and setter methods for the **message** field.

#### 2. Return JSON Response:

- Change the return type for **getHello()** method to **HelloResponse**
- Inside the **getHello** method, create an instance of **HelloResponse** with the message "Hello, World!" and return it.

#### 3. Testing:

- Use Postman to send an HTTP GET request to your service endpoint (**/hello**).
- Verify that the response is in JSON format and contains the expected greeting message.

## EXERCISE 2

In this exercise you will improve the previous JAX-RS example by taking the following steps:

### COPY THE PREVIOUS PROJECT

Please copy the previous project by doing right-click on the project and select copy. Usually, a copy of the project will be created with the same name. If happens, please do right-click again on the project and select rename and then please provide a new name for that like **HelloWorld2**.

### CREATE A CLASS CALLED USER

Create a class named **User** with three attributes: **id**, **name** and **email**. Please also add a default constructor and also one with all three attributes. Finally, add getters and setters.

### UPDATE HELLO WORLD RESOURCE CLASS

Please completely revise the **HelloWorldResource** class by taking the following steps:

#### 1. Initialize Users:

- In the **HelloWorldResource** class, create a static block to initialize a list of users. You can create a List and call it *users*.
- Add at least three users to the list with unique IDs, names, and email addresses.

#### 2. GET Method to Retrieve Users:

- Implement a method named **getUsers** in the **HelloWorldResource** class. The return type for the method must be **List<user>**.
- Annotate the method with **@GET** and **@Produces(MediaType.APPLICATION\_JSON)**.
- Inside the method, return the list of users.

### RUN THE APPLICATION AND TEST IT USING POSTMAN

Use Postman to send an HTTP GET request to your service endpoint (**/hello**). Verify that the response is in JSON format and contains the expected list of users.

## EXERCISE 3

In this exercise, you will improve the second exercise by doing the following steps:

### COPY THE PREVIOUS PROJECT

Please copy the previous project by doing right-click on the project and select copy. Usually, a copy of the project will be created with the same name. If happens, please do right-click again on the project and select rename and then please provide a new name for that like **HelloWorld3**.

### UPDATE USER CLASS

- Change the fields to private fields for **userId** (integer) and **userName** (String).
- Create a constructor that takes parameters for **userId** and **userName** and initializes the fields.
- Generate getter and setter methods for **userId** and **userName**.

### UPDATE HELLO WORLD RESOURCE CLASS

#### 1. Initialize Users:

- In the **HelloWorldResource** class, create a static block to initialize a map of users. This time, you need to use **Map** as a data structure.
- Add at least three users to the map with unique IDs and names.

#### 2. GET Method to Retrieve a User by ID:

- Implement a method named **getUserById** inside the **HelloWorldResource** class.
- Annotate **getUserById** method with **@GET**, **@Path("/user/{userId}")**, and **@Produces(MediaType.APPLICATION\_JSON)**.
- Use the **@PathParam** annotation to extract the **userId** from the URL.
- Check if the user exists in the map and return a **User** instance with the user's information.

#### 3. GET Method to Retrieve All Users:

- Annotate **getAllUsers** the method with **@GET** and **@Path("/allUsers")**.
- The return type for the method should be **Map<Integer, String>**
- Return the entire map of users.

#### 4. **Testing:**

- Create a simple test class or use a tool like Postman to send HTTP GET requests to your service endpoints (**/hello/user/{userId}** and **/hello/allUsers**).
- Verify that the responses contain the expected user information.