

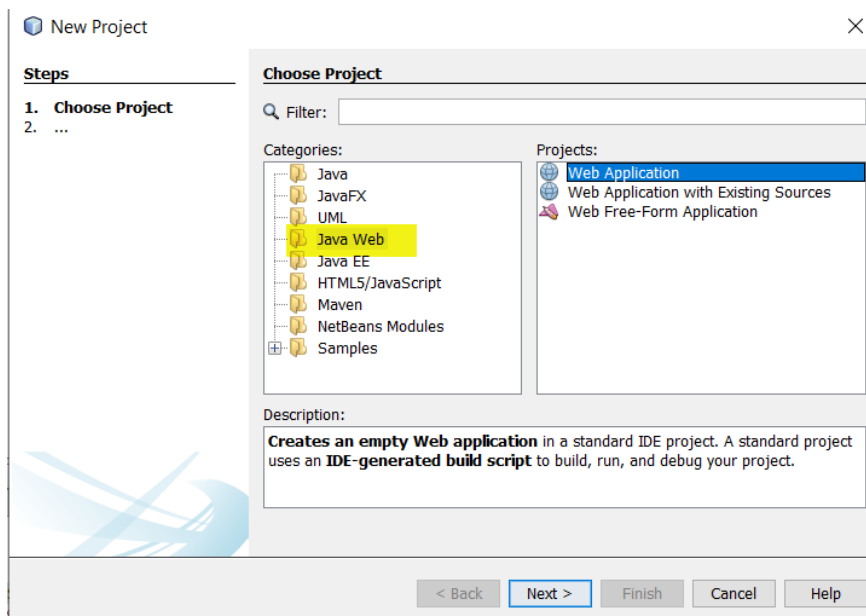
Tutorial – Using EJB as a Jax-RS Web service

In this tutorial we'll be developing an extremely simple web service to cater a hypothetical scenario. What we will be developing is a simple service that takes in a vehicles registration number and based on the first few characters (CAA,NA,PD etc..) determine vehicle category and inform the end user the category of driver's license needed to drive that vehicle. For simplicities sake and since the point of the exercise is merely to further explain underlying concepts, we will not be using any database connections etc. and will purely use hardcoded values.

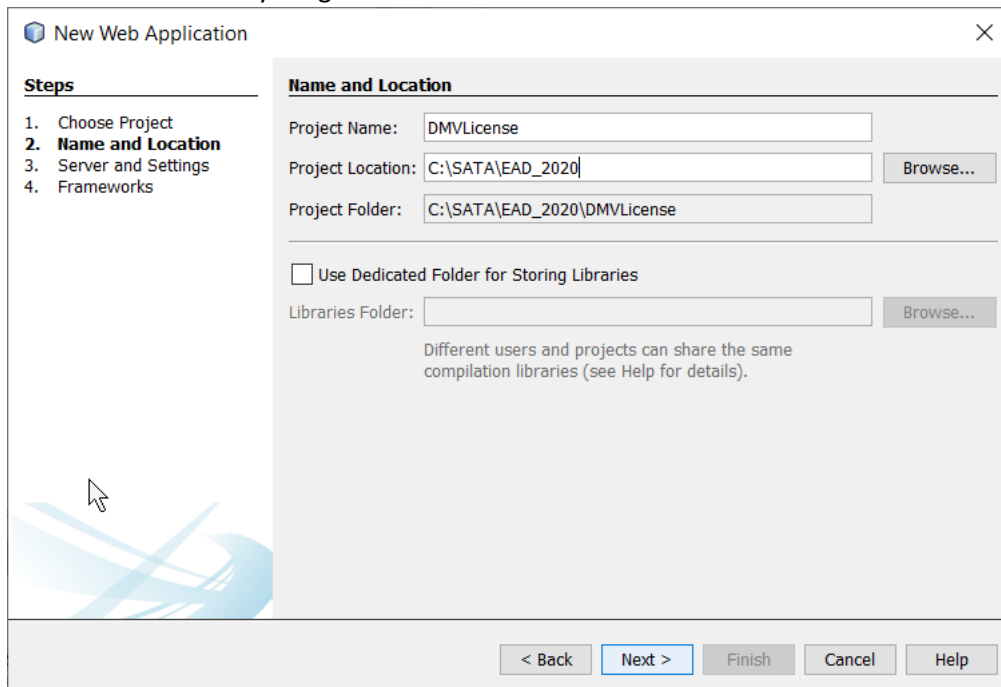
Perquisites : Netbeans 8.2, Postman.

The first step is to fire up Netbeans IDE 8.2

Create a new project – type should be web application.



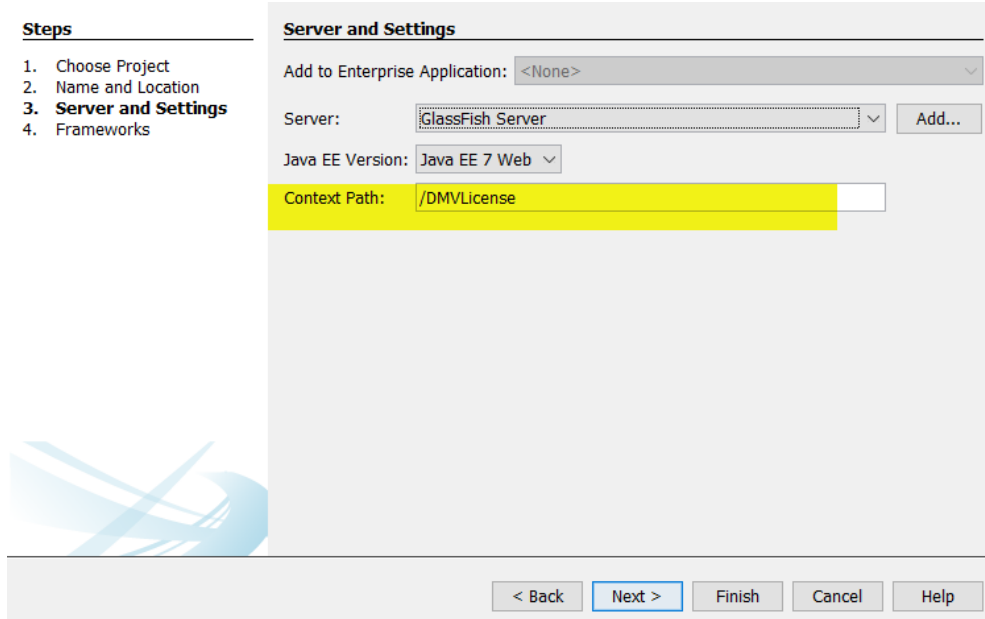
We can name this anything



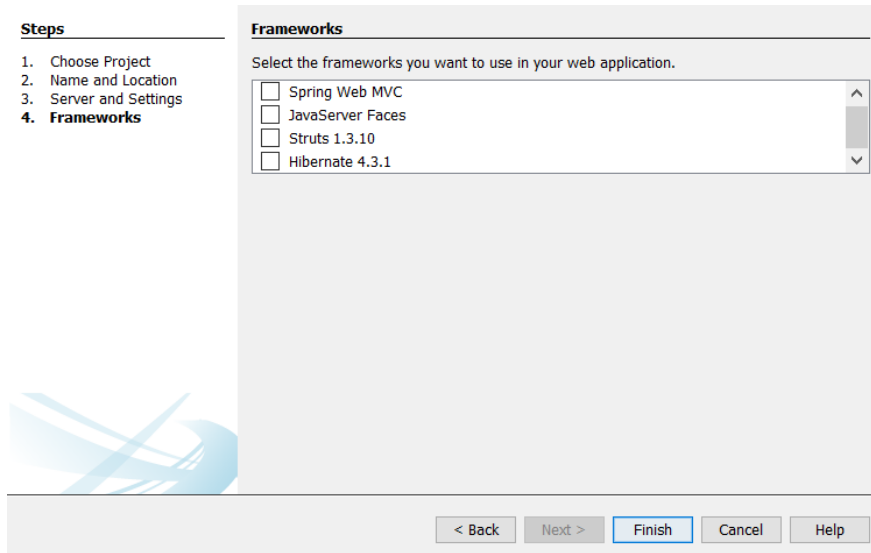
The screenshot shows the 'New Web Application' wizard window. On the left, a 'Steps' list shows: 1. Choose Project, 2. **Name and Location**, 3. Server and Settings, 4. Frameworks. The main area is titled 'Name and Location' and contains the following fields: 'Project Name' with the value 'DMVLicense', 'Project Location' with the value 'C:\SATA\EAD_2020' and a 'Browse...' button, and 'Project Folder' with the value 'C:\SATA\EAD_2020\DMVLicense'. Below these is a checkbox 'Use Dedicated Folder for Storing Libraries' which is unchecked, followed by a 'Libraries Folder' field and a 'Browse...' button. A note states: 'Different users and projects can share the same compilation libraries (see Help for details)'. At the bottom are buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Pay attention to the context path -this will be part of the URL of the service.

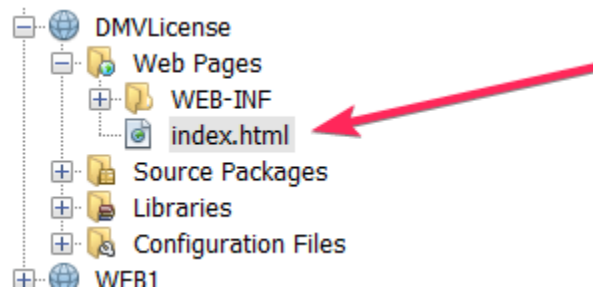
Keep the defaults for the next few steps in the Wizard.



The screenshot shows the 'New Web Application' wizard window at Step 3: 'Server and Settings'. The 'Steps' list on the left shows: 1. Choose Project, 2. Name and Location, 3. **Server and Settings**, 4. Frameworks. The main area contains: 'Add to Enterprise Application:' with a dropdown set to '<None>', 'Server:' with a dropdown set to 'GlassFish Server' and an 'Add...' button, 'Java EE Version:' with a dropdown set to 'Java EE 7 Web', and 'Context Path:' with the value '/DMVLicense' highlighted in yellow. At the bottom are buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.



Once the project is created locate the index.html file where we will be adding a text box and a submit button.



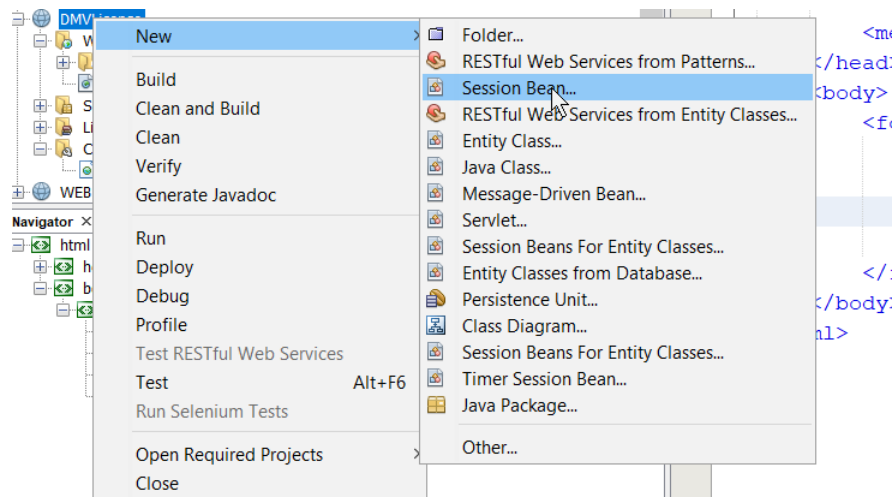
The following html shows the creation of a simple text box and a submit button that we will be using to submit the Vehicle registration number.

```
<html>
  <head>
    <title>DMV License Category Service</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  </head>
  <body>
    <form action="/DMVLicense/webresources/scanplate"
method="POST">
      Model <input name="regnum" value="CAA1234">
      <br>
      <input type="submit">
    </form>
```

```
</body>
</html>
```

Vehicle Registration Number :

Once the simple index html page is done, we will create a new session bean – on the project node select new Session bean. Give it a name – it's not important what you call it. For clarities sake we can keep this in a package called 'beans'



New Session Bean

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

EJB Name: PlateNumberManager

Project: DMVLicense

Location: Source Packages

Package: beans

Session Type:

☒ Stateless

☐ Stateful

☐ Singleton

Create Interface:

☐ Local

< Back Next > **Finish** Cancel Help

The session bean will be kept as stateless. Inside the session bean we'll be adding the business method/ business logic – the following method accepts the registration number of the vehicle and based on the first character decides what category it is and what license type is required to drive it.

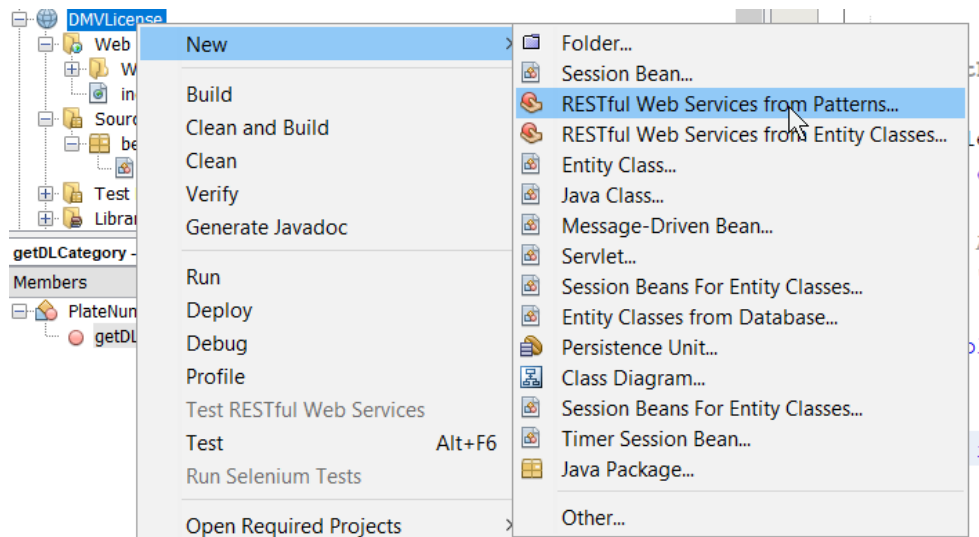
```
public String getDLCategory(String regnum)
{
    String dlcategory="Unknown Licence Category";
    switch(Character.toUpperCase(regnum.charAt(0))) {
        case 'C':
        case 'K':
        case 'P':
        case 'D':
            dlcategory="Light Vehicle";
            break;
        case 'L':
        case 'N':
            dlcategory="Heavy Vehicle";
            break;
        case 'M':
        case 'B':
            dlcategory="Motor Bicycle";
            break;
        case 'Y':
        case 'A':
            dlcategory="Three Wheeler";
            break;
    }
    return dlcategory;
}
```

```
@Stateless
public class PlateNumberManager {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")

    public String getDLCategory(String regnum)
    {
        String dlcategory="Unknown Licence Category";
        switch(Character.toUpperCase(regnum.charAt(0))) {
            case 'C':
            case 'K':
            case 'P':
            case 'D':
                dlcategory="Light Vehicle";
                break;
            case 'L':
            case 'N':
                dlcategory="Heavy Vehicle";
                break;
            case 'M':
            case 'B':
                dlcategory="Motor Bicycle";
                break;
        }
    }
}
```

Next we can add a new Restful web service.



Steps

1. Choose File Type
- 2. Select Pattern**
3. Specify Resource Classes

Select Pattern

Select a RESTful web service design pattern:

- ☒ Simple Root Resource
- ☐ Container-Item
- ☐ Client-Controlled Container-Item

Description:

Create a RESTful root resource class with GET and PUT methods using Java API for RESTful Web Service (JSR-311). This pattern is useful for creating a simple HelloWorld service and wrapper services for invoking WSDL-based web services.

On the next page you will be specifying class name, URI, and representation type of the resource.

< Back

Next >

Finish

Cancel

Help

We will use a new package called services to keep this – we can give a path (this will be appended to the service URL) the class name can be anything basically. Change the MIME type to text.html.

Steps

1. Choose File Type
2. Select Pattern
- 3. Specify Resource Classes**

Specify Resource Classes

Project: DMVLicense

Location: Source Packages

Resource Package: services

Path: scanplate

Class Name: AnalyzeNumber

MIME Type: text/html

Representation Class: java.lang.String

Select...

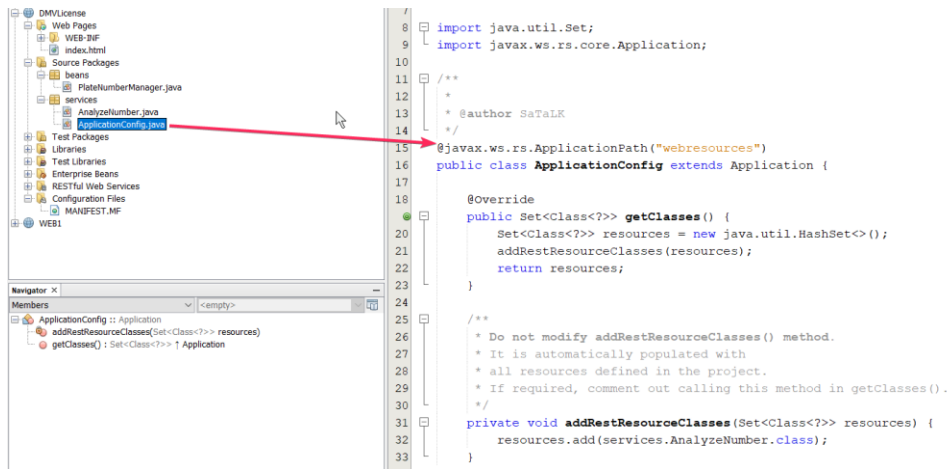
< Back

Next >

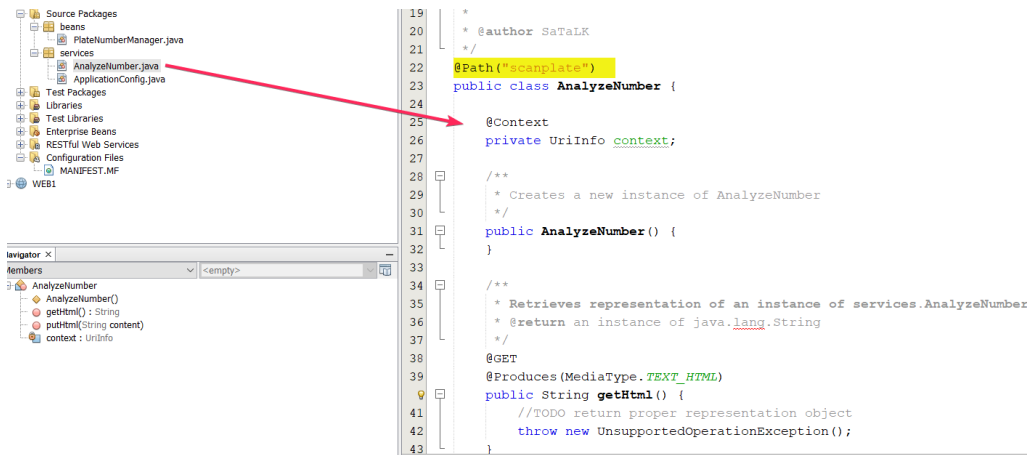
Finish

Cancel

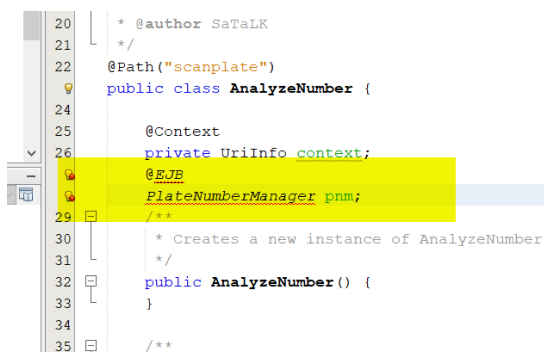
Help



Two new classes are created – the application config class will have some information pertaining to the path etc. also observe the path you provided in the previous dialog box is appended to the @Path annotation in the AnalyzeNumber class we created.



The annotated path will direct us to this class and depending on the action defined in the index.html page it will call the relevant method with either @GET or @POST Annotations



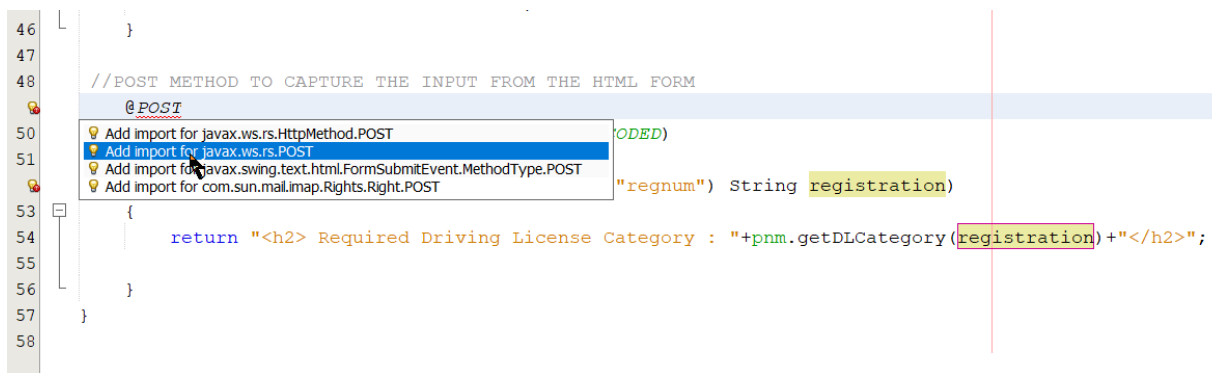
Since we will be calling the business method in the beans class PlateNumberManager we will need to create an instance of that class with the @EJB annotation

```
@EJB
    PlateNumberManager pnm;
```

Now since we defined the action as 'POST' in the index.html we will get rid of the @PUT annotated method and instead write a method with @POST annotation

```
@POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.TEXT_HTML)
    public String getCategoryPost(@FormParam("regnum") String
registration)
    {
        return "<h2> Required Driving License Category :
"+pnm.getDLCategory(registration)+"</h2>";
    }
```

Make sure to fix the imports as follows



The screenshot shows an IDE with a code editor. Line 50 has a suggestion popup for imports. The code is as follows:

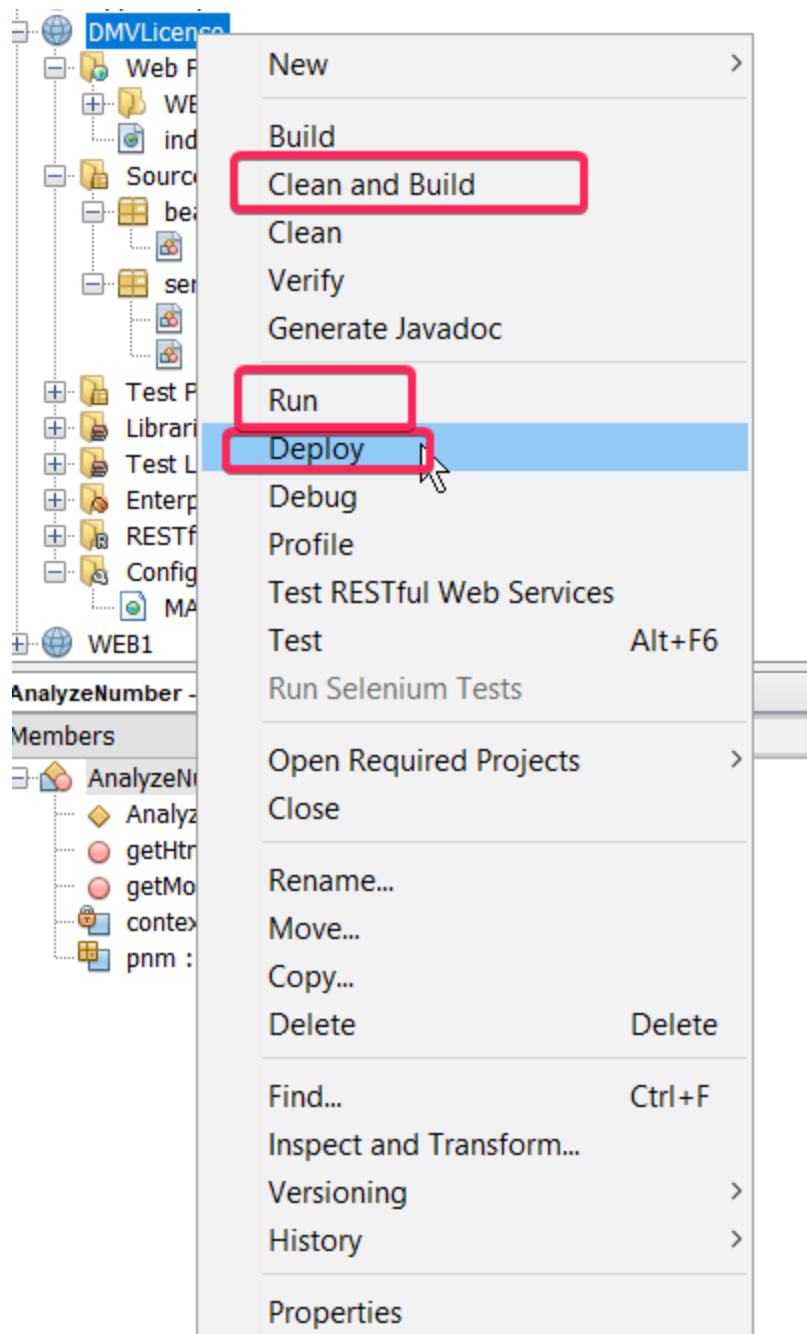
```
46     }
47
48     //POST METHOD TO CAPTURE THE INPUT FROM THE HTML FORM
49     @POST
50     // Add import for javax.ws.rs.HttpMethod.POST (JDK 8.0.0_102)
51     // Add import for javax.ws.rs.POST (JDK 8.0.0_102)
52     // Add import for javax.swing.text.html.FormSubmitEvent.MethodType.POST (JDK 8.0.0_102)
53     // Add import for com.sun.mail.imap.Rights.Right.POST (JDK 8.0.0_102)
54     @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
55     @Produces(MediaType.TEXT_HTML)
56     public String getCategoryPost(@FormParam("regnum") String registration)
57     {
58         return "<h2> Required Driving License Category : "+pnm.getDLCategory(registration)+"</h2>";
59     }
```

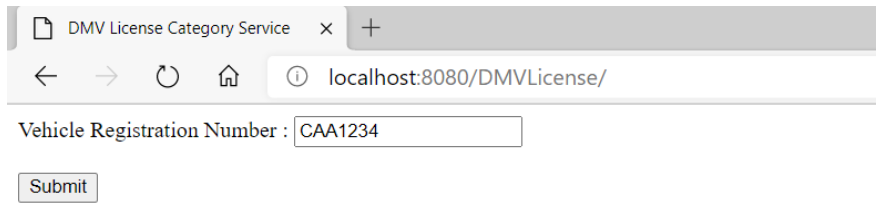
The full code block is shown below:

```
//POST METHOD TO CAPTURE THE INPUT FROM THE HTML FORM
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.TEXT_HTML)
public String getCategoryPost(@FormParam("regnum") String registration)
{
    return "<h2> Required Driving License Category : "+pnm.getDLCategory(registration)+"</h2>";
}
```

Now we're basically calling the post action and invoking the beans Business method.

Click on your project clean and build and then 'Run'

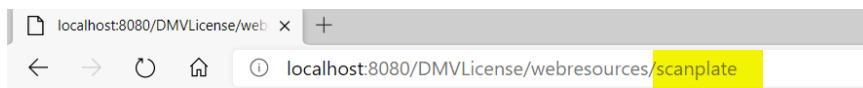




DMV License Category Service

← → ↻ 🏠 ⓘ localhost:8080/DMVLICENSE/

Vehicle Registration Number :



Required Driving License Category : Light Vehicle

Next up we can explore other options of passing parameters.

We can start by getting the registration number as a query parameter

```
@GET
@Path("getcategory")
@Produces(MediaType.TEXT_HTML)
public String
getCategoryParams(@QueryParam("regnum") @DefaultValue("CBE1234")
String registration){
    return "<h2> Required Driving License Category :
"+pnm.getDLCategory(registration)+"</h2>";
}
```

```

@GET

@Path("getcategory")

@Produces(MediaType.TEXT_HTML)

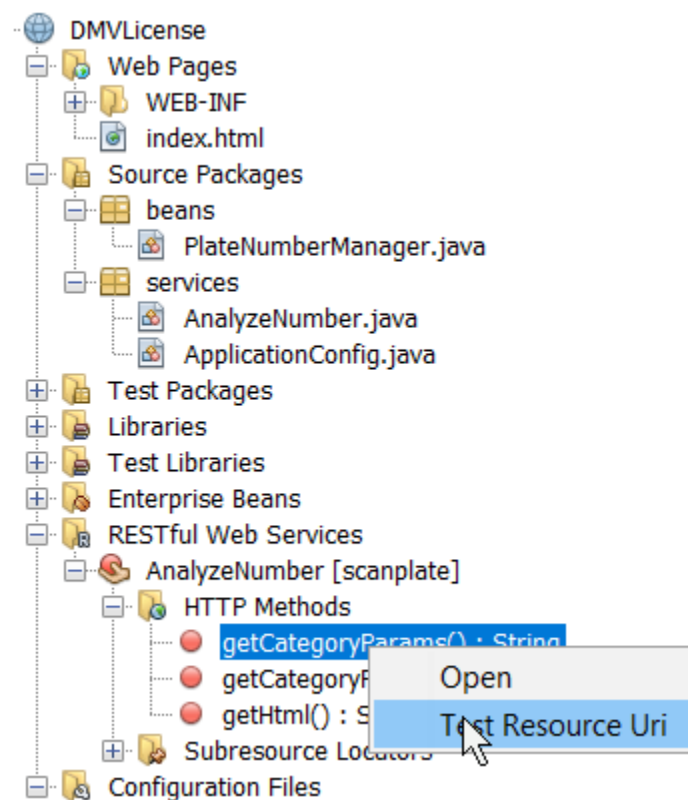
public String
getCategoryParams(@QueryParam("regnum")@DefaultValue("CBE1234") String
registration){

    return "<h2> Required Driving License Category :
"+pnm.getDLCategory(registration)+"</h2>";

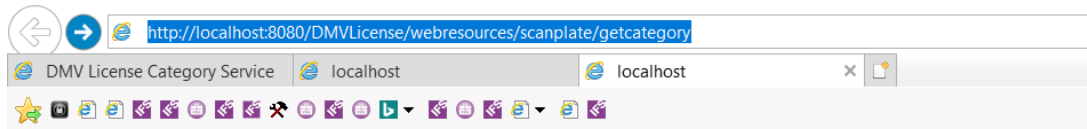
}

```

Notice that since this is a 'GET' method NetBeans allows us to test this method via the navigator



This will call the method via the browser – but without query parameters so it will run with the default value we provided.

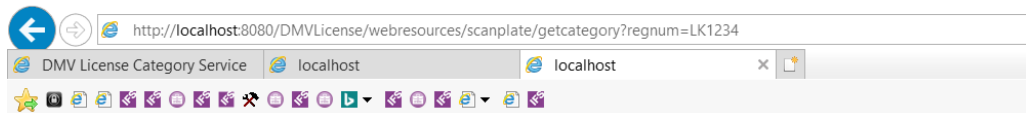


Required Driving License Category : Light Vehicle

to pass in the query parameter we can modify this URL to below

<http://localhost:8080/DMVLICENSE/webresources/scanplate/getcategory?regnum=LK1234>

note that we need to provide the correct query parameter name (in our case regnum)



Required Driving License Category : Heavy Vehicle

As per our logic L falls into the 'Heavy Vehicle Category' and is correctly fetched.

One more option we can try out is to use path variables

Add the following method to the AnalyzeNumber.java file

```
@GET
@Path("/{regnum}")
@Produces(MediaType.TEXT_HTML)
public String getCategoryPath(@PathParam("regnum") String
registration)
{
    return "<h2> Required Driving License Category :
"+pnm.getDLCategory(registration)+"</h2>";
}
```

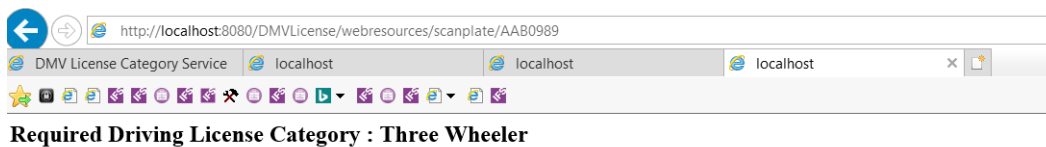
```

    }
    //GET METHOD WITH PATH VARIABLES
    @GET
    @Path("{regnum}")
    @Produces(MediaType.TEXT_HTML)
    public String getCategoryPath(@PathParam("regnum") String registration)
    {
        return "<h2> Required Driving License Category : "+pnm.getDLCategory(registration)+"</h2>";
    }
}

```

To invoke this method, we need to append the query parameter into the URL as follows

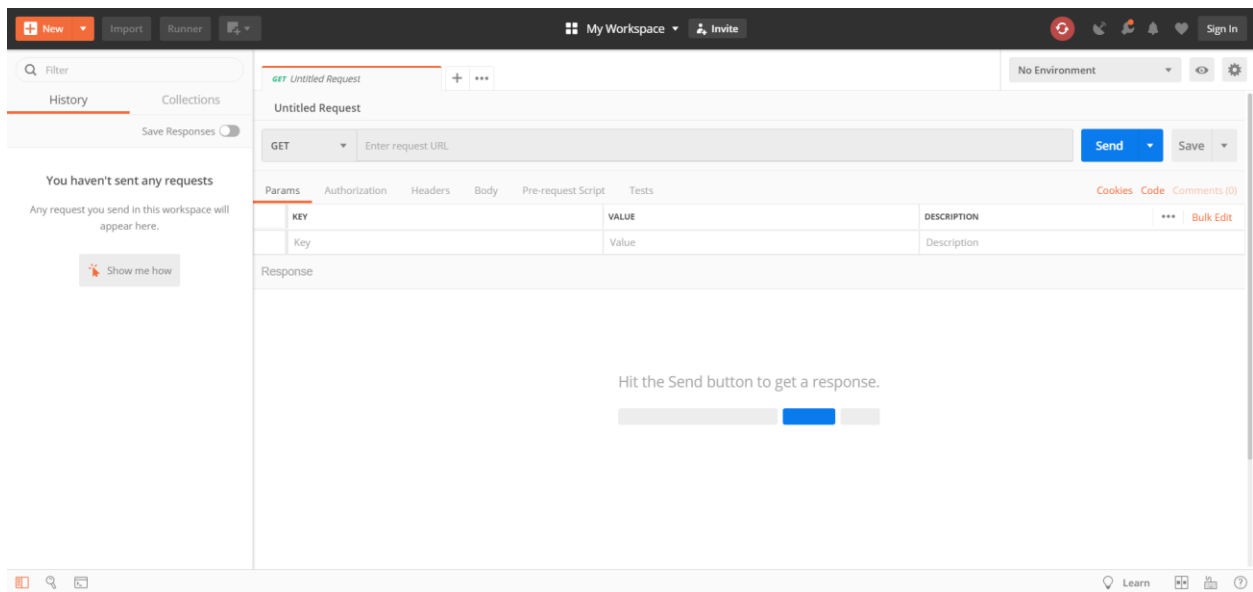
<http://localhost:8080/DMVLicense/webresources/scanplate/AAB0989>



Calling the webservices using POSTMAN

Download postman from <https://www.postman.com/>

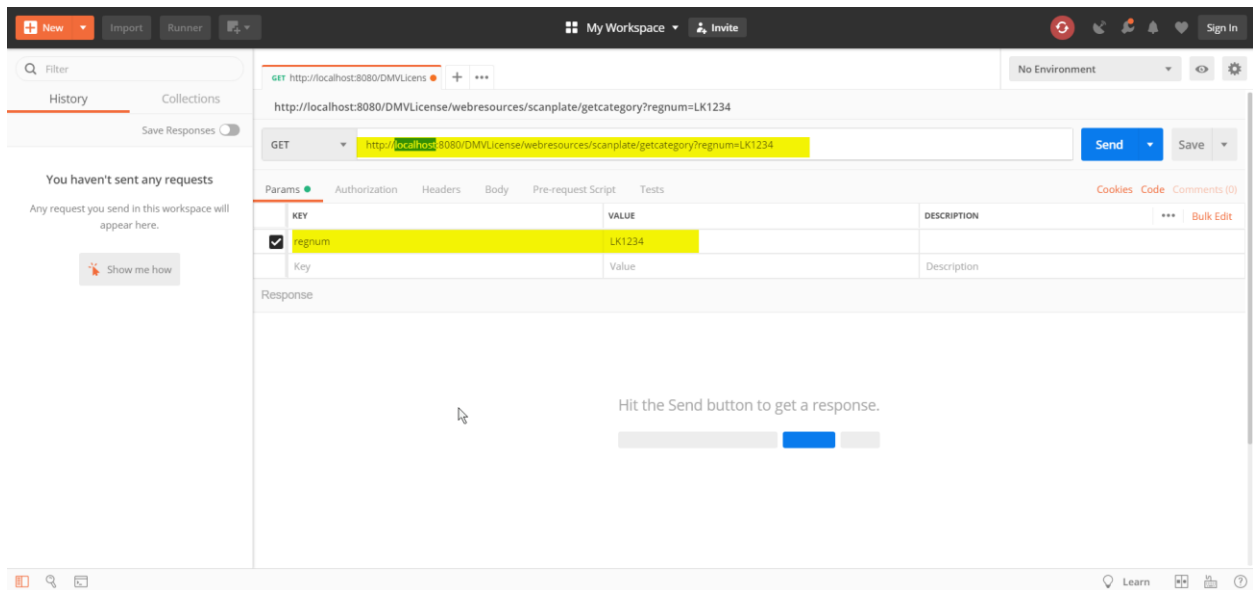
Once installed and when you open the application on the start page you can create a new Get Request



Simply copy the URL with the query parameter.

<http://localhost:8080/DMVLICENSE/webresources/scanplate/getcategory?regnum=LK1234>

postman automatically detects the query param.



Hit the send button and you should be able to get a response.

GET http://localhost:8080/DMVLicens + ... No Environment

http://localhost:8080/DMVLicence/webresources/scanplate/getcategory?regnum=LK1234

GET http://localhost:8080/DMVLicence/webresources/scanplate/getcategory?regnum=LK1234 Send

Params Authorization Headers Body Pre-request Script Tests Cookies

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> regnum	LK1234	
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 59 ms Size: 32

Pretty Raw Preview HTML

```
i 1 <h2> Required Driving License Category : Heavy Vehicle</h2>
```

The text is returned with the HTML tags so if you want to see the HTML preview you can click on the 'Preview' button.

GET http://localhost:8080/DMVLicens + ...

http://localhost:8080/DMVLicence/webresources/scanplate/getcategory?regnum=LK1234

GET http://localhost:8080/DMVLicence/webresources/scanplate/getcategory?regnum=LK1234

Params Authorization Headers Body Pre-request Script Tests

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> regnum	LK1234	
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 C

Pretty Raw Preview

Required Driving License Category : Heavy Vehicle

Similarly, postman will read XML/JSON etc and that can be set by the MediaType on the GetMethod in the restful web service.

You can also try the request with path variable on Postman too ...

<http://localhost:8080/DMVLicense/webresources/scanplate/AAB0989>