Usar JSP JavaScript Template Literal hoy por hoy suele ser muy necesario. ¿Quien trabaja hoy en día con JSP? . Parece que todo el mundo usa motores de plantillas o tecnologías como Angular y React a la hora de abordar los proyectos. Puede ser cierto… pero la realidad es tozuda y mucha gente se encuentra con la necesidad de actualizar relativamente esos proyectos que tienen construidos con JSP y que en su día introdujeron JQuery o otra tecnología de Javascript para enriquecerlo y que el proyecto pudiera vivir unos años más. ¿Podemos usar en un proyecto clásico de JSP características modernas de programación de JavaScript como aquellas que vienen en JavaScript ES6?. La respuesta es que SI , pero que en algunas ocasiones hay que retocar alguna cosa. Vamos a hablar un poco de JSP JavaScript Template Literal o cómo usar motores de plantillas de JavaScript ES6 en un aplicación antigua de JSP. Para ello utilizaremos un Servlet que nos devuelva información en formato JSON al cliente.

```
package com.arquitecturajava;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.gson.Gson;

/**
 * Servlet implementation class ServletJSON
 */
```

```
@WebServlet("/ServletJSON")
public class ServletJSON extends HttpServlet {
  private static final long serialVersionUID = 1L;

  protected void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException {
    // TODO Auto-generated method stub

    Persona p1 = new Persona("pedro", "perez", 20);
    Persona p2 = new Persona("pedro", "perez", 20);
    List<Persona> lista = new ArrayList<Persona>();
    lista.add(p1);
    lista.add(p2);

    PrintWriter pw = response.getWriter();
    Gson gson = new Gson();
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    pw.println(gson.toJson(lista));
  }

}
```

Este Servlet publicará un listado de Personas en formato JSON en su URL.

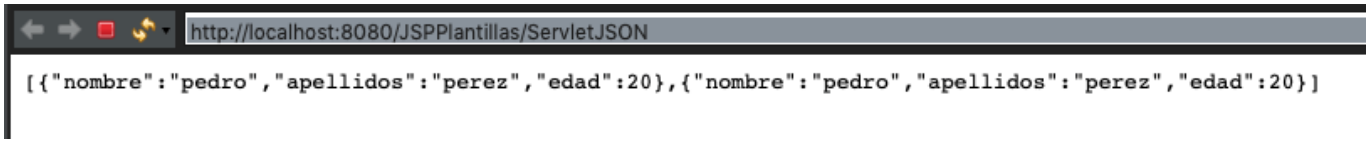
```
package com.arquitecturajava;

public class Persona {

  private String nombre;
  private String apellidos;
```

```
  private int edad;
  public String getNombre() {
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  public String getApellidos() {
    return apellidos;
  }
  public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
  }
  public int getEdad() {
    return edad;
  }
  public void setEdad(int edad) {
    this.edad = edad;
  }
  public Persona(String nombre, String apellidos, int edad) {
    super();
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.edad = edad;
  }
  public Persona() {
    super();
  }
}
```

Si solicitamos la url al servidor nos mostrará los datos sin ningún tipo de problema.



JSP JavaScript Template Literal

Nos queda de ver cómo invocamos desde JavaScript esta url y mostramos una lista de elementos con el contenido de los datos en JSON. Para ello vamos a construir un pequeño script con jquery.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>

<meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript" src="jquery-3.4.0.js"></script>
<script type="text/javascript">

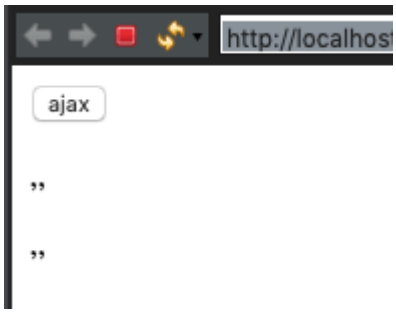
$(document).ready(function() {

  $("#ajax").click(function(){
    $.get("ServletJSON",function(datos){
      datos.forEach(function(persona) {
$("body").append(`<p>${persona.nombre},${persona.apellidos},${persona.
edad}</p>`);
      })
    });
  });
```

```
})


</script>
</head>
<body>
<input type="button" value="ajax" id="ajax"/>
</body>
</html>
```

En principio no hace falta hacer más , pero sorprendentemente nos encontramos con el siguiente resultado:



Da la sensación de que no podemos integrar Javascript Template Literals o que el navegador no lo soporta. La realidad es bien distinta recordemos que a nivel de JSP existe lo que se denomina el expresion language es decir yo puedo mostrar una variable escribiendo:

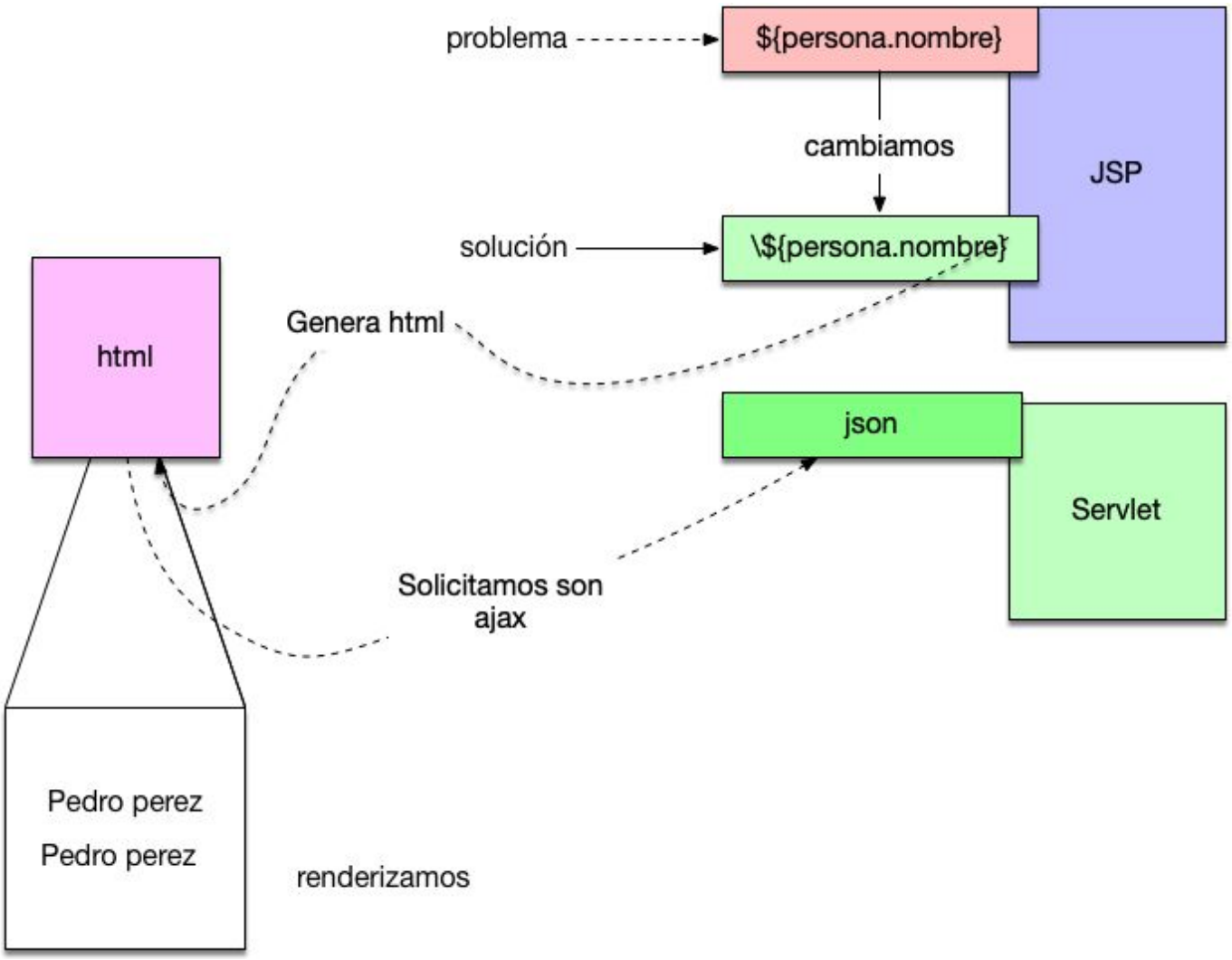
${persona.nombre}

Es aquí donde tenemos el problema ya que los JavaScript Template Literals funcionan de la misma forma a la hora de definir una variable.

${persona.nombre}

¿Como podemos solventar este problema? . Muy sencillo deberemos obligar al servidor a

escapar estas variables para que puedan llegar al cliente como tales y que este al hacer la petición ajax sea capaz de usarlas.



Vamos a verlo en código:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript" src="jquery-3.4.0.js"></script>
<script type="text/javascript">

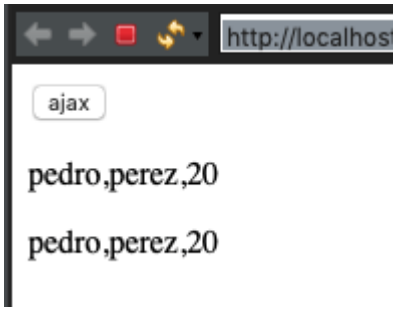
$(document).ready(function() {

  $("#ajax").click(function(){
    $.get("ServletJSON",function(datos){
      datos.forEach(function(persona) {
$("body").append(`<p>\${persona.nombre},\${persona.apellidos},\${perso
na.edad}</p>`);
      })
    });
  });
})



</script>
</head>
<body>
<input type="button" value="ajax" id="ajax"/>
</body>
</html>
```

Fijemonos como en vez de usar ${persona.nombre} hemos introducido una barra \${persona.nombre} . De esta forma ya no tendremos ningún problema y si pedimos



Acabamos de integrar tecnologías muy antiguas con tecnologías modernas , este es un ejemplo del uso de JSP JavaScript Template Literals.

1. JavaScript Template for loop y como usarlo
2. JavaScript Arrow Function y scopes
3. El concepto de JavaScript Function Composition
4. JavaScript Template Literal Mozilla