



Ciclo 3

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Lectura 2 - Integración entre aplicaciones Frontend y Backend:
JSP – Servlets y SpringBoot

| Integración entre aplicaciones Frontend y Backend: JSP – Servlets y SpringBoot

Programación lado Frontend

Dado que ya tenemos en las dos semanas anteriores los componentes básicos del frontend (JSP), y de backend (SpringBoot), es momento de realizar la integración de los dos componentes, para lo cual habremos de conocer los elementos activos de Java que se colocan incrustados dentro de las páginas JSP. Se hace a continuación una relación de estos elementos a saber:

- **Scriptlets JSP:** son partes de código Java que se colocan dentro de tags o etiquetas en una página JSP como estas: `<% instrucciones; %>`
- **Expresiones JSP:** son las variables (o atributos Java) cuyos valores se muestran en la página. Se utilizan las siguientes etiquetas: `<% =variable %>`
- **Declaraciones JSP:** corresponden a la declaración de métodos Java dentro de una página JSP. Se utilizan las siguientes etiquetas: `<%! Definición método() {} %>`



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Un ejemplo de las tres declaraciones anteriores, dentro de una página JSP puede ser el siguiente:

```

<%@page import="java.time.Month"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <!-- Expression tag example -->
    Converting a string to uppercase:
    <%=new String("Hello World").toUpperCase()%>

    <!-- JSP Scriptlet Example -->
    <%
        for (Month month : Month.values()) {
            out.println(month);
        }
    %>

    <!-- JSP Declaration Tag Example -->
    <%!int cube(int n) {
        return n * n * n;
    }%>
    <%= "Cube of 3 is:" + cube(3)%>

</body>
</html>

```

JSP expression tag with syntax <%= expression %>

JSP scriptlet tag with syntax <% statements; %>

JSP declaration tag with syntax <%! Declaration %>

Tomado de: <https://www.javaguides.net/2019/01/jsp-scripting-elements.html>

De la misma forma, se recuerda una instrucción Java muy utilizada dentro de Servlets y páginas JSP, la cual es `request.getParameter(<nombre>)`; que permiten obtener los parámetros que se envían a través de las páginas JSP a los Servlets, y viceversa (recuerden la semana 2, para obtener la cédula que la página "hola.JSP" envía al servlet).

```

<form method="get" action="./DemoServlet">
    <table>
        <tr>
            <td><label>Nombre:</label></td>
            <td><input type="text" name="nombre"></td>
        <tr>
            <td><label>Cédula:</label></td>
            <td><input type="text" name="cedula"></td>
        </tr>
    </table>
</form>

```



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Y en el Servlet:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) {  
    // TODO Auto-generated method stub  
    response.getWriter().append("Served at: ").append(request.getContextPath());  
    String nombre = request.getParameter("nombre");  
    String cedula = request.getParameter("cedula");  
}
```

Para importar clases dentro de páginas JSP, se utilizan las **directivas de página JSP**, cuya sintaxis es la siguiente, mediante `<%@ page import='librería'%>`. Un ejemplo es el siguiente:

```
<%@ page import='java.util.Date'%>  
<%@ page import='co.edu.unbosque.ciclo3appFrontEnd.model.Usuarios'%>  
<%@ page import='java.util.ArrayList'%>
```

Para poner en práctica estos conceptos, vamos a retomar a nuestro proyecto de la semana 2, cuyo Java Project lo denominamos “ciclo3demo”, con el fin de modificar y agregar el código necesario para que pueda ingresarse los datos completos del usuario a la base de datos, y recuperarse todos los datos de usuarios ingresados. Para lo anterior, vamos a modificar la página JSP denominada “hola.jsp”, para que luzca de esta forma.

CREACION DE USUARIOS

| | |
|--|--------------------------|
| Cedula: | <input type="text"/> |
| Nombre: | <input type="text"/> |
| Correo Electronico: | <input type="text"/> |
| Usuario: | <input type="text"/> |
| Password: | <input type="password"/> |
| <input type="button" value="Agregar"/> | |
| <input type="button" value="Listar Usuarios"/> | |



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Igualmente, a esta página (`hola.jsp`) la vamos a renombrar a "`inicio.jsp`". Como se puede apreciar es un diseño HTML muy básico, por lo cual, nos concentraremos en la funcionalidad esencial, dejando el diseño y los detalles a cada grupo. El código HTML de la página es el siguiente:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Tienda Generica</title>
</head>
<body>
<p align="center"> CREACION DE USUARIOS</p>
<form method="get" action="./inicio">
<table>
<tr>
<td><label>Cedula:</label></td>
<td><input type="text" name="cedula"></td>
</tr>
<tr>
<td><label>Nombre:</label></td>
<td><input type="text" name="nombre"></td>
</tr>
<tr>
<td><label>Correo Electronico:</label></td>
<td><input type="text" name="email"></td>
</tr>
<tr>
<td><label>Usuario:</label></td>
<td><input type="text" name="usuario"></td>
</tr>
<tr>
<td><label>Password:</label></td>
<td><input type="text" name="password"></td>
</tr>
<tr>
<td><input type="submit" value="Agregar" name="Agregar"></td>
</tr>
<tr>
<td><button type="submit" name="Listar">Listar Usuarios</button></td>
</tr>
</table>
</form>
</body>
</html>
```

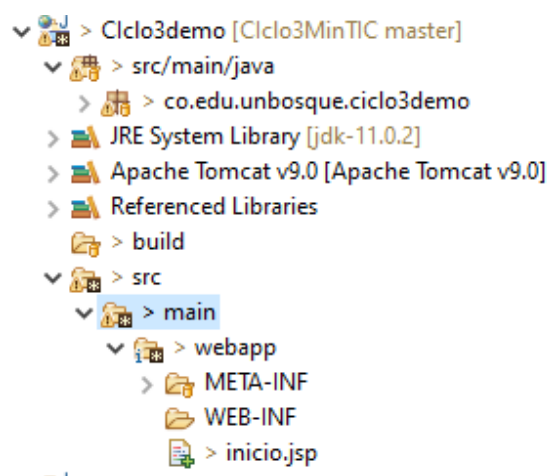


Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Nótese que la acción a realizar por GET de la página, ahora apunta a la URL relativa “/inicio”, la cual redirigirá hacia el Servlet “`Demoservlet.java`”. Igualmente, se modificó un botón para que se pudiera agregar el registro de usuario ingresado con sus respectivos campos, y se agregó otro, que corresponde a listar usuarios.

Finalmente, la estructura del proyecto Java, con la página web JSP renombrada a “`inicio.jsp`”, queda de la siguiente forma:



Codificación Java para Envío y Recepción de Mensajes JSON

Ahora, debemos abordar la manera de escribir código Java que nos permita recibir y generar un mensaje JSON desde y hacia las APIs de **usuarios** del backend que ya construimos en la semana 3, de la cual, la especificación de los mensajes JSON para la agregación de usuarios, y para listar todos los usuarios es la siguiente (extraído del documento de descripción del proyecto)



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Para agregar usuarios, el mensaje es el siguiente:

POST /usuarios/guardar guardar

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|-----------|--|-------------|----------------|--|
| usuario | (required) <div><div></div></div> <div>Parameter content type: application/json</div> | usuario | body | <div>Model</div> <div>Example Value</div> <pre>{ "cedula_usuario": 0, "email_usuario": "string", "nombre_usuario": "string", "password": "string", "usuario": "string" }</pre> |

El método para consumir la API es POST, con el sufijo /usuarios/guardar, de lo que se conformaría la URL <http://localhost:8080/usuarios/guardar> (solo es un ejemplo), y el formato de mensaje JSON, está en la parte amarilla a la derecha. La ampliamos aquí:

```
{
  "cedula_usuario": 0,
  "email_usuario": "string",
  "nombre_usuario": "string",
  "password": "string",
  "usuario": "string"
}
```

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Como se puede apreciar, los campos JSON tienen, a excepción de la cédula del cliente (que es un dato de tipo Integer, y es la llave primaria (PK) de consulta, las cuales concuerdan con la definición de la clase **Usuarios** del proyecto Java hecho en la semana 3 “ciclo3back”. Este es un ejemplo del mensaje JSON:

```
{
  "cedula_usuario": 194938,
  "nombre_usuario": "Jose Camargo",
  "email_usuario": "chepe@tienda.com",
  "usuario": "chepe",
  "password": "chepito"
},
```

Para listar usuarios, la especificación de la API es la siguiente:

GET /usuarios/listar

Response Class (Status 200)
OK

Model

Example Value

```
[
  {
    "cedula_usuario": 0,
    "email_usuario": "string",
    "nombre_usuario": "string",
    "password": "string",
    "usuario": "string"
  }
]
```




Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

El método para consumir la API es GET, con el sufijo /usuarios/listar, de lo que se conformaría la URL <http://localhost:8080/usuarios/listar>, de lo que, el mensaje JSON resultado por cada usuario es igual que el visto en la API de agregar usuarios, sin embargo, esta API puede retornar más de un usuario, por lo que puede tener este tipo de respuesta, en donde se ven dos (2) usuarios, Jose Camargo, y Jorge González:

```
[
  {
    "cedula_usuario": 194938,
    "nombre_usuario": "Jose Camargo",
    "email_usuario": "chepe@tienda.com",
    "usuario": "chepe",
    "password": "chepito"
  },
  {
    "cedula_usuario": 432233,
    "nombre_usuario": "Jorge Gonzalez",
    "email_usuario": "jgonzo@algo.com",
    "usuario": "jgonzo",
    "password": "alguito"
  }
]
```

Una vez conocida esta especificación, la deberemos utilizar para escribir algunos métodos en Java para el consumo de esta API, para lo cual nos vamos a apoyar en una librería Java de terceros que nos permita generar y recibir mensajes JSON. Esta librería se denomina “json-simple”, la cual la podremos descargar de esta URL:

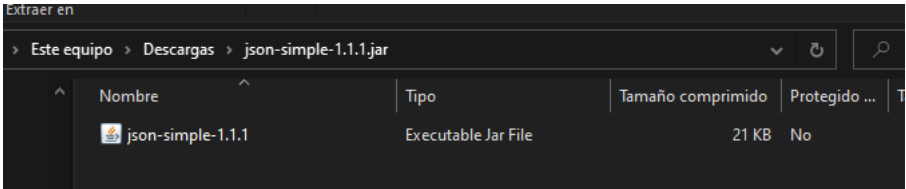
<http://www.java2s.com/Code/Jar/j/Downloadjsonsimple111jar.htm>.

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.



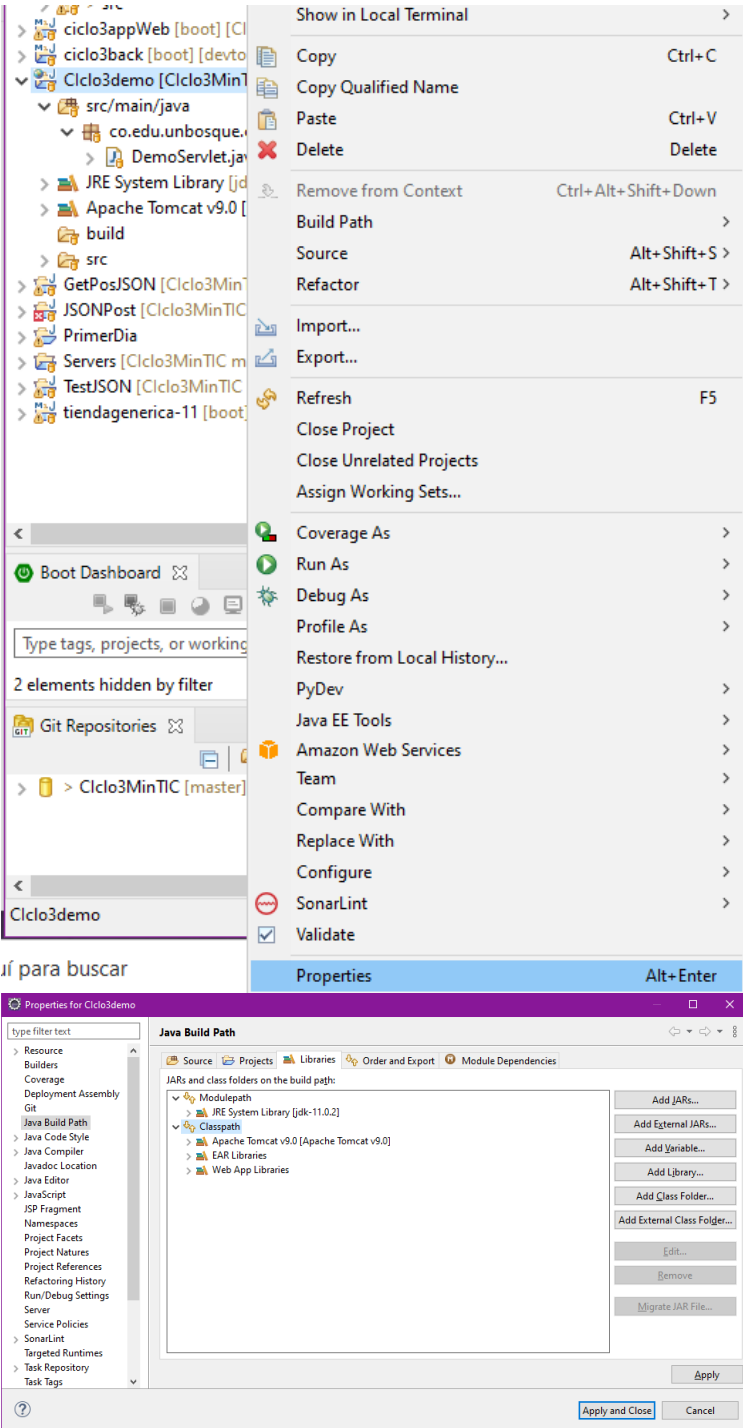
Una vez se haya descargado el archivo .ZIP, se descomprime, y debería quedar un archivo .Jar como este:



Es importante recordar la carpeta donde ubicamos el archivo, porque vamos a referirnos a esta al momento de configurar el proyecto Java “ciclo3demo”, hecho en la semana 2 para agregar esta librería en Eclipse tanto en el build path, como en el deployment assembly. En el proyecto “ciclo3demo” hacemos clic derecho en la carpeta principal, y hacemos click en properties, y, finalmente, hacemos clic en la parte izquierda donde dice “Java build Path”.

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

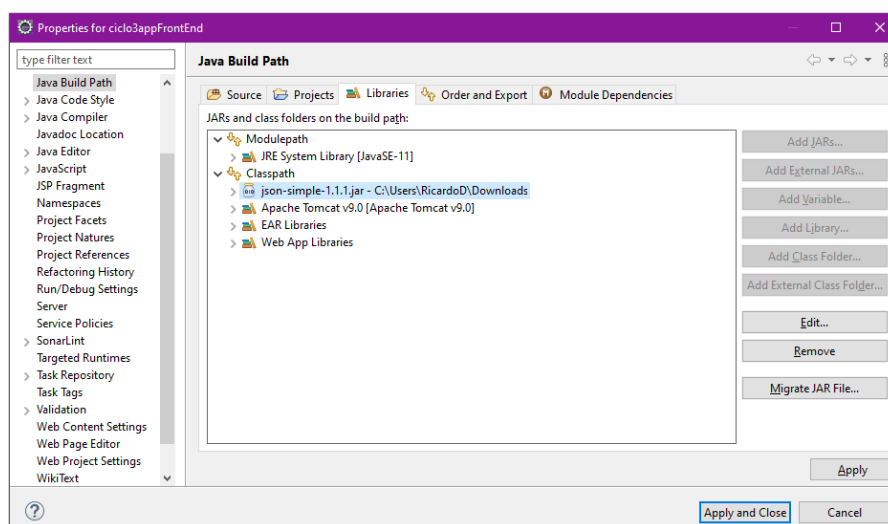




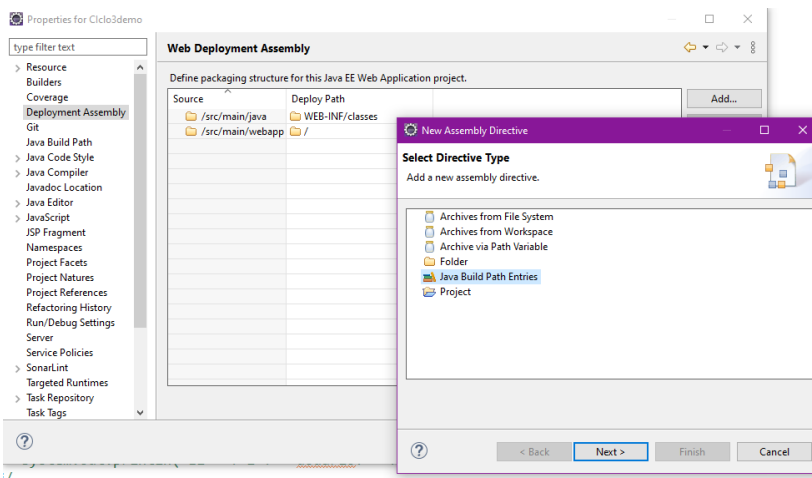
Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Dentro de la opción “java build path”, en la ventana a la derecha hacemos clic en la pestaña “libraries”, donde están los proyectos (o librerías) requeridos para la construcción de nuestro proyecto. Hacemos clic en “Classpath”, y, y clic en el botón “Add External JARs”, y se nos muestra una ventana, en donde buscamos la ubicación del archivo “json-simple-1.1.1.jar” que ya descargamos anteriormente, finalmente, hacemos clic en Abrir, y nos debe queda la librería colocada de esta forma:



Posteriormente, en la misma ventana, en la parte izquierda de Properties, hacemos clic en la opción “Deployment Assembly”, hacemos clic en “Add”, y seleccionamos “Java Build Path Entries”, como se ve abajo.

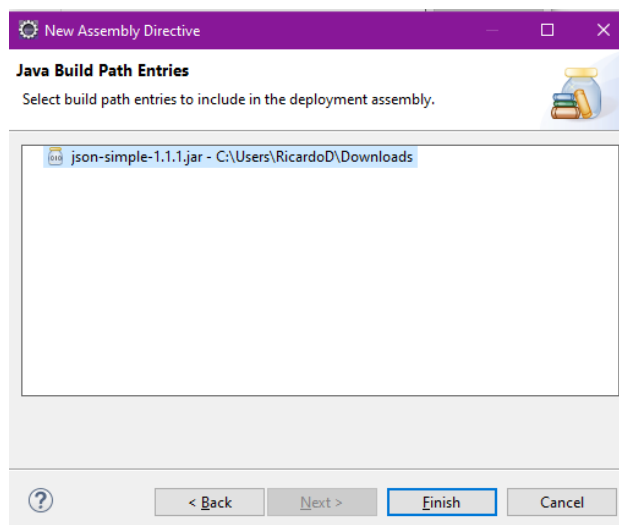




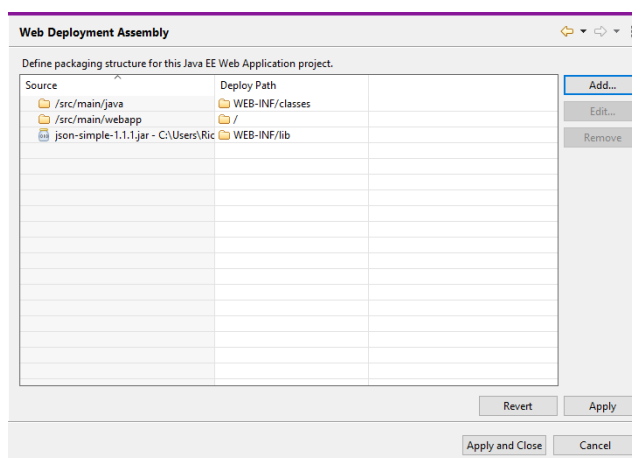
Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Hacemos clic en “Next”, y nos aparece la librería JSON que ya cargamos en el build path.



Hacemos clic en ella, y hacemos clic en “Finish”. Nos debe quedar instalada de la siguiente forma:



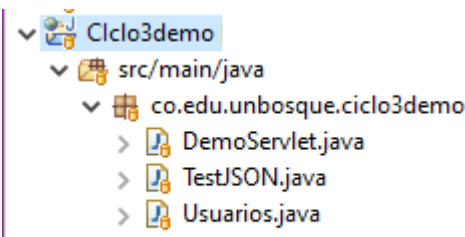
Finalizamos todo el proceso haciendo clic en “Apply and Close”.

Vamos a hacer uso de la librería Simple JSON mediante una nueva clase Java, que llamaremos `TestJSON.java` en el package `co.edu.unbosque.ciclo3demo`. Vamos a escribir tres (3) métodos (dos principales, y uno auxiliar) que nos permitirán insertar un nuevo usuario y

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

recuperar los usuarios, por medio de una operación GET y operación POST a las respectivas APIs diseñadas en la semana 3, que ya describimos arriba. Igualmente, vamos a copiar la clase Java Usuarios del proyecto Java “ciclo3back” que hicimos en la semana 3. Deberemos tener una estructura en el proyecto Java de esta forma:



En la clase `TestJSON.java`, vamos a escribir los siguientes métodos:

- a) `parsingUsuarios`: este método recibe un dato de tipo `String` correspondiente al mensaje JSON recibido de la API de listar usuarios (recordar ejemplo de JSON de arriba), y devuelve un `ArrayList` de tipo `Usuarios` con los usuarios encontrados en el JSON.

```

public static ArrayList<Usuarios> parsingUsuarios(String json) throws ParseException {
    JSONParser jsonParser = new JSONParser();
    ArrayList<Usuarios> lista = new ArrayList<Usuarios>();
    JSONArray usuarios = (JSONArray) jsonParser.parse(json);
    Iterator i = usuarios.iterator();
    while (i.hasNext()) {
        JSONObject innerObj = (JSONObject) i.next();
        Usuarios usuario = new Usuarios();
        usuario.setCedula_usuario(innerObj.get("cedula_usuario").toString());
        usuario.setEmail_usuario(innerObj.get("email_usuario").toString());
        usuario.setNombre_usuario(innerObj.get("nombre_usuario").toString());
        usuario.setPassword(innerObj.get("password").toString());
        usuario.setUsuario(innerObj.get("usuario").toString());
        lista.add(usuario);
    }
    return lista;
}

```




Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

- b) `getJSON`: este método realiza la operación GET desde la URL de la API para listar todos los usuarios, y obtiene el mensaje JSON, el cual es procesado en por el método `parsingUsuarios()`, y devuelve un `ArrayList` de tipo `Usuarios` con los usuarios encontrados en el JSON

```
public static ArrayList<Usuarios> getJSON() throws IOException, ParseException{
    url = new URL(sitio+"usuarios/listar");
    HttpURLConnection http = (HttpURLConnection)url.openConnection();
    http.setRequestMethod("GET");
    http.setRequestProperty("Accept", "application/json");
    InputStream respuesta = http.getInputStream();
    byte[] inp = respuesta.readAllBytes();
    String json = "";
    for (int i = 0; i<inp.length ; i++) {
        json += (char)inp[i];
    }
    ArrayList<Usuarios> lista = new ArrayList<Usuarios>();
    lista = parsingUsuarios(json);
    http.disconnect();
    return lista;
}
```

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

- c) `postJSON`: este método realiza la operación POST a la URL de la API para agregar un nuevo usuario. Recibe como parámetro un objeto de tipo `Usuario` (con el usuario que se quiere agregar), y retorna una respuesta HTTP de la URL: si es exitosa la operación, será 200, de lo contrario, serán respuestas (401, 403, y 404).

```

public static int postJSON(Usuarios usuario) throws IOException {
    url = new URL(sitio+"usuarios/guardar");

    HttpURLConnection http;
    http = (HttpURLConnection)url.openConnection();
    try {
        http.setRequestMethod("POST");
    } catch (ProtocolException e) {
        e.printStackTrace();
    }
    http.setDoOutput(true);
    http.setRequestProperty("Accept", "application/json");
    http.setRequestProperty("Content-Type", "application/json");
    String data = "{"
        + "\"cedula_usuario\": \""+ usuario.getCedula_usuario()
        + "\", \"email_usuario\": \""+usuario.getEmail_usuario()
        + "\", \"nombre_usuario\": \""+usuario.getNombre_usuario()
        + "\", \"password\": \""+usuario.getPassword()
        + "\", \"usuario\": \""+usuario.getUsuario()
        + "\"}";
    byte[] out = data.getBytes(StandardCharsets.UTF_8);
    OutputStream stream = http.getOutputStream();
    stream.write(out);
    int respuesta = http.getResponseCode();
    http.disconnect();
    return respuesta;
}

```

Durante el proceso de escritura de estos tres métodos indicados, saldrán errores por la no aplicación de ciertas librerías, ejemplo: `ArrayList`, `IOException`, `URL`, etc., para solventar esto, presionamos las teclas `<ctrl>+<shift>+<o>`., y Eclipse incluirá en los import las librerías necesarias.

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Agregamos como atributos de la clase `TestJSON`, los siguientes:

```
private static URL url;
private static String sitio = "http://localhost:8080/";
```

Y, finalmente, deberemos tener la clase `TestJSON`, con una estructura como la que se muestra en la parte de abajo, así:

```
package co.edu.unbosque.ciclo3demo;

import java.io.IOException;

public class TestJSON {

    private static URL url;
    private static String sitio = "http://localhost:8080/";

    public static ArrayList<Usuarios> getJSON() throws IOException, ParseException{

    }

    public static ArrayList<Usuarios> parsingUsuarios(String json) throws ParseException {

    }

    public static int postJSON(Usuarios usuario) throws IOException {

    }
}
```

Codificación Java en `DemoServlet.java` para incorporar los métodos JSON.

Ahora nos aprestamos a incluir dentro de la clase `DemoServlet.java`, lo correspondiente a invocar los métodos para realizar el ingreso de un nuevo usuario, y de listar todos los usuarios de la tabla **usuarios**. Lo anterior lo realizaremos, mediante la modificación del método `doGet()` que hicimos en la semana 2, el cual era este:

```
30 protected void doGet(HttpServletRequest request, HttpServletResponse response) th
31     // TODO Auto-generated method stub
32     response.getWriter().append("Served at: ").append(request.getContextPath());
33     String nombre = request.getParameter("nombre");
34     String cedula = request.getParameter("cedula");
35     PrintWriter writer = response.getWriter();
36     if( nombre != null && cedula != null) {
37         writer.println("Bienvenido "+nombre+" a mi JSP");
38     }
39     else
40         writer.println("Error: Nombre o cédula faltante!");
41     writer.close();
42 }
```

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

Dado que, en el archivo JSP modificado “inicio.JSP”, están todos los campos correspondientes a la tabla **usuarios**, debemos modificar los request.getParameter , para que, en primer lugar, se procese la acción del usuario al presionar cada botón del formulario, y, dependiendo de cuál fue, se deberá proceder con una acción distinta (agregar usuario, listar usuarios).

El código “inicio.jsp”, con estos primeros cambios, quedaría de esta forma:

```

30 protected void doGet(HttpServletRequest request, HttpServletResponse response)
31     throws ServletException, IOException {
32     // TODO Auto-generated method stub
33     response.getWriter().append("Served at: ").append(request.getContextPath());
34     String listar = request.getParameter("Listar");
35     String agregar = request.getParameter("Agregar");
36     if (agregar != null) {
37
38         //aqui van las instrucciones Java para agregar un usuario
39
40     }
41
42     if (listar != null) {
43
44         //aqui van las instrucciones Java para listar los usuarios
45
46     }
47
48 }

```

Ahora, se está evaluando si la acción que recibe de “inicio.JSP”, corresponde el parámetro “Listar”, o “Agregar” – esto se sabe si la variable tiene un valor cuando es presionado el botón, o es null. Se indica por medio de dos condicionales (if) el bloque de instrucciones a realizar según la acción.

Para la opción Agregar, vamos a escribir un método que se llame agregarUsuario(), el cual inicializará un objeto de tipo Usuarios con los request.getParameter de todos los atributos, e invocará el método postJSON, de la clase TestJSON, ya configurada, con dicho objeto. Finalmente, obtendrá una respuesta HTTP desde la API /usuarios/guardar del backend. Si la respuesta es 200 (o sea, HTTP OK), dará un mensaje por la página de éxito, en caso contrario, indicará el error.

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

```

public void agregarUsuario(HttpServletRequest request, HttpServletResponse response) {
    Usuarios usuario = new Usuarios();
    usuario.setNombre_usuario(request.getParameter("nombre"));
    usuario.setCedula_usuario(request.getParameter("cedula"));
    usuario.setEmail_usuario(request.getParameter("email"));
    usuario.setUsuario(request.getParameter("usuario"));
    usuario.setPassword(request.getParameter("password"));
    int respuesta = 0;
    try {
        respuesta = TestJSON.postJSON(usuario);
        PrintWriter writer = response.getWriter();
        if (respuesta == 200) {
            writer.println("Registro Agregado!");
        } else {
            writer.println("Error: " + respuesta);
        }
        writer.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

Para la opción Listar, vamos a escribir el método de nombre `listarUsuarios()`. Para este método, debemos invocar el método `getJSON()`, ya definido en la clase `TestJSON`, y guardar la lista de usuarios en una variable de tipo `ArrayList` que llamaremos `lista`.

A continuación, viene una parte nueva de la aplicación, la cual, va a utilizar una nueva página JSP que crearemos en el siguiente paso con nombre `resultado.jsp`. A esta nueva página enviaremos los resultados de la variable `lista`, por medio de la instrucción `request.setAttribute`, la cual asigna un atributo (o variable) para sea enviada (operación forward) a la página JSP que creamos (por medio de instanciar un objeto de la clase `RequestDispatcher`), y se pueda mostrar en formato HTML, por medio de la escritura de un *scriptlet* que diseñaremos posteriormente. El método `listarUsuarios()`, entonces, quedaría así:

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

```

public void listarUsuarios(HttpServletRequest request, HttpServletResponse response) {
    try {
        ArrayList<Usuarios> lista = TestJSON.getJSON();
        String pagina = "/resultado.jsp";
        request.setAttribute("lista", lista);
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(pagina);
        dispatcher.forward(request, response);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

Finalmente, se agregan los dos métodos dentro de las dos condicionales indicadas en el método `doGet()`, quedando así.

Codificación del *Scriptlet* Java en la página `resultado.jsp` para interactuar con el Servlet.

Nos resta ahora, la creación de la página JSP con nombre `resultado.jsp`. El proceso para crearla es el mismo que utilizamos en la semana 2 para crear la primera. Cuando la tengamos,

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    response.getWriter().append("Served at: ").append(request.getContextPath());
    String listar = request.getParameter("Listar");
    String agregar = request.getParameter("Agregar");
    if (agregar != null) {
        agregarUsuario(request, response);
    }

    if (listar != null) {
        listarUsuarios(request, response);
    }
}

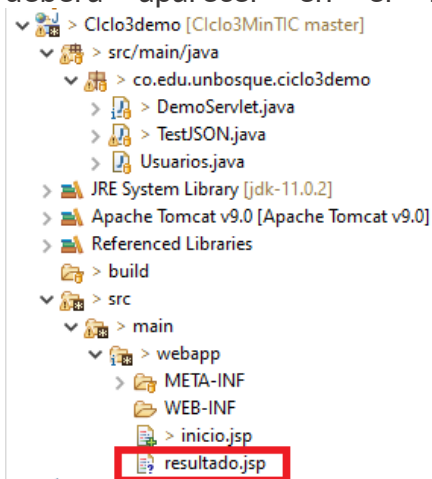
```




Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

deberá aparecer en el Project Explorer de Eclipse de la siguiente forma:



Debemos a continuación, construir una tabla con las etiquetas `<table></table>`, para mostrar los resultados enviados por el Servlet en forma tabular. Esta tabla tendrá dos partes: a) los encabezados normales, esto es: que solo será etiquetas y texto HTML, y b) los registros de usuarios serán escritos con un *Scriptlet* Java. La parte a) la escribiremos de esta forma:

```
<html>
<head>
<meta charset="ISO-8859-1">
<title>Tienda Generica</title>
</head>
<body>
  <p align="center">
    LISTADO DE USUARIOS</p>
  <p align="center">
    Hora servidor es<%=new Date()%></p>
  <table>
    <tr>
      <td><label>Cedula</label></td>
      <td><label>Nombre</label></td>
      <td><label>Correo</label></td>
      <td><label>Usuario</label></td>
      <td><label>Password</label></td>
    </tr>
```

Nótese que esta parte está incompleta (no cierran etiquetas `<table>`, ni `<body>`).

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.

La parte b) la haremos con instrucciones Java que nos permita recibir el atributo `lista`, enviado por el Servlet, por medio de la instrucción `request.getAttribute()`, y convirtiéndola (mediante casting explícito) al tipo de dato `ArrayList` de la clase `Usuarios`, luego, haremos el recorrido de esta lista, mediante un `for`, y tomando cada variable, para mostrarse entre las etiquetas de las tablas `<td></td>`, por medio de expresiones JSP. Esto se puede escribir de la siguiente forma:

```
<%
ArrayList<Usuarios> lista = (ArrayList<Usuarios>) request.getAttribute("lista");
for (Usuarios usuario : lista) {
%>
<tr>
<td><%=usuario.getCedula_usuario()%></td>
<td><%=usuario.getNombre_usuario()%></td>
<td><%=usuario.getEmail_usuario()%></td>
<td><%=usuario.getUsuario()%></td>
<td><%=usuario.getPassword()%></td>
</tr>
<%
}
%>
```



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Incorporando las partes a) y b) ya descritas, tendremos la sección <body> de la página de la siguiente forma:

```
<body>
  <p align="center">
    LISTADO DE USUARIOS</p>
  <p align="center">
    Hora servidor es<%=new Date()%></p>
  <table>
    <tr>
      <td><label>Cedula</label></td>
      <td><label>Nombre</label></td>
      <td><label>Correo</label></td>
      <td><label>Usuario</label></td>
      <td><label>Password</label></td>
    </tr>

    <%
      ArrayList<Usuarios> lista = (ArrayList<Usuarios>) request.getAttribute("lista");
      for (Usuarios usuario : lista) {
        %>
        <tr>
          <td><%=usuario.getCedula_usuario()%></td>
          <td><%=usuario.getNombre_usuario()%></td>
          <td><%=usuario.getEmail_usuario()%></td>
          <td><%=usuario.getUsuario()%></td>
          <td><%=usuario.getPassword()%></td>
        </tr>
        <%
      }
    %>
  </table>
</body>
```

Eclipse marca errores en ciertas partes del código escrito, lo cual se hizo así a propósito para denotar la necesidad que ahora tenemos de importar las librerías java necesarias para que sean reconocidas en esta página JSP, mediante directivas de páginas JSP, las cuales se ubican antes de la etiqueta <html>, y quedarían de la siguiente forma:

```
<!DOCTYPE html>
<%@ page import='java.util.Date'%>
<%@ page import='co.edu.unbosque.ciclo3demo.Usuarios'%>
<%@ page import='java.util.ArrayList'%>
<html>
```



Semana 4

Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Finalmente, la página `resultado.jsp`, quedaría de la siguiente forma:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<%@ page import='java.util.Date'%>
<%@ page import='co.edu.unbosque.ciclo3demo.Usuarios'%>
<%@ page import='java.util.ArrayList'%>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Tienda Generica</title>
</head>
<body>
<p align="center">
    LISTADO DE USUARIOS</p>
<p align="center">
    Hora servidor es<%=new Date()%></p>
<table>
<tr>
<td><label>Cedula</label></td>
<td><label>Nombre</label></td>
<td><label>Correo</label></td>
<td><label>Usuario</label></td>
<td><label>Password</label></td>
</tr>

<%
    ArrayList<Usuarios> lista = (ArrayList<Usuarios>) request.getAttribute("lista");
    for (Usuarios usuario : lista) {
    %>
<tr>
<td><%=usuario.getCedula_usuario()%></td>
<td><%=usuario.getNombre_usuario()%></td>
<td><%=usuario.getEmail_usuario()%></td>
<td><%=usuario.getUsuario()%></td>
<td><%=usuario.getPassword()%></td>
</tr>
<%
    }
    %>
</table>
</body>
</html>
```

Prueba de integración frontend - backend.

Es el momento de realizar la prueba de integración de las dos aplicaciones: frontend y backend, para lo cual, deberemos realizar un pequeño cambio en la configuración en el backend del puerto HTTP por donde escuchará las peticiones, ya que ambas aplicaciones por defecto están escuchando el puerto 8080, lo cual hace que ambas aplicaciones entren en conflicto.



Semana 4

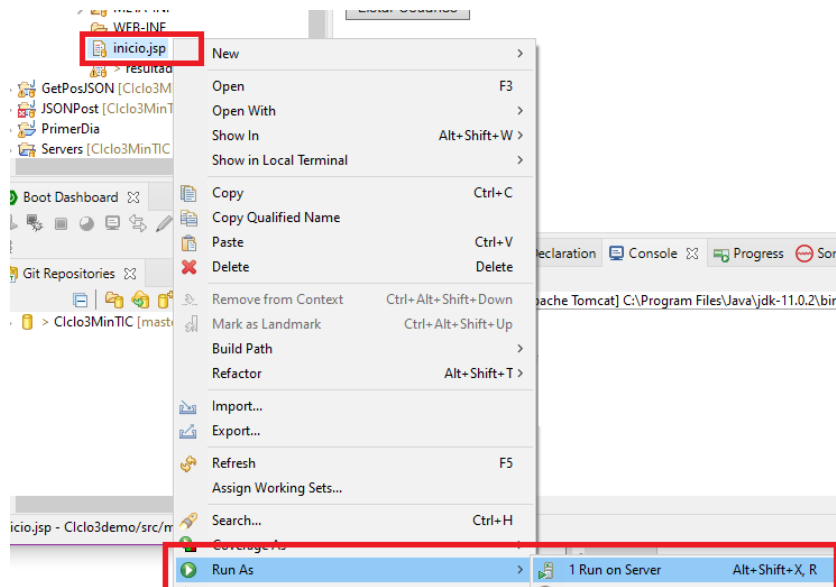
Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

Vamos a agregar en el archivo de configuración de `system.properties`, la siguiente instrucción:

```
server.port = 5000
```

Con lo anterior, ahora nuestra aplicación de backend estará escuchando peticiones por el puerto 5000 (se coloca este puerto para que sea compatible con las especificaciones de Amazon Web Services al momento de subir esta aplicación al Cloud – lo cual es requerimiento de AWS). Con este cambio, ejecutaremos la aplicación de backend, como ya aprendimos en la semana 3.

Ahora es turno para ejecutar la aplicación de frontend, según las instrucciones de la semana 2. Para efectos de que ejecutemos la página `inicio.jsp`, lo haremos de esta forma: Hacemos clic derecho sobre la página `inicio.jsp` en el Package Explorer, y ejecutamos Run As -> Run on Server.



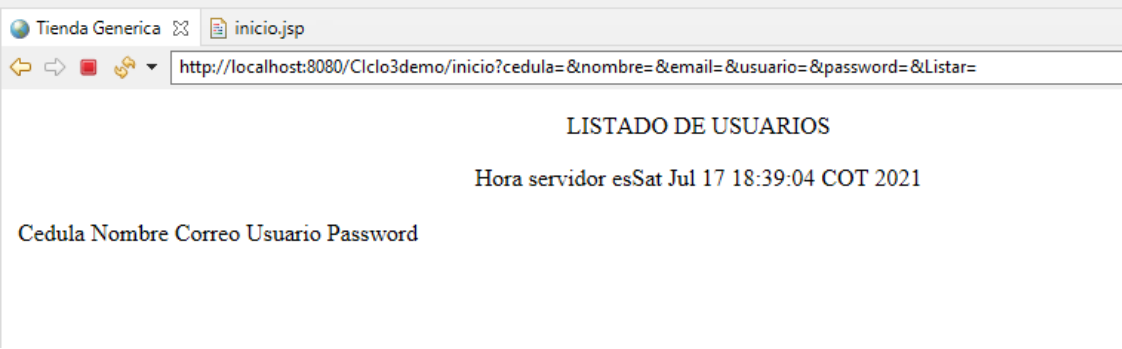
Y en la pantalla siguiente hacemos clic en Finish, y dentro de Eclipse se abre una página Web como la de la imagen de abajo (aclaramos que esta es otra manera de ejecutar una aplicación Web, adicional a la que vimos en la semana 2).

Semana 4

Arquitectura de software parte 2 y metodología de desarrollo Scrum parte 2.



Si presionamos Listar Usuarios, no debemos obtener ningún registro como tal, de lo que veremos este resultado.



Regresamos con la fecha “Atrás”



Y procedemos a agregar un registro de un usuario como este, y presionamos “Agregar”.

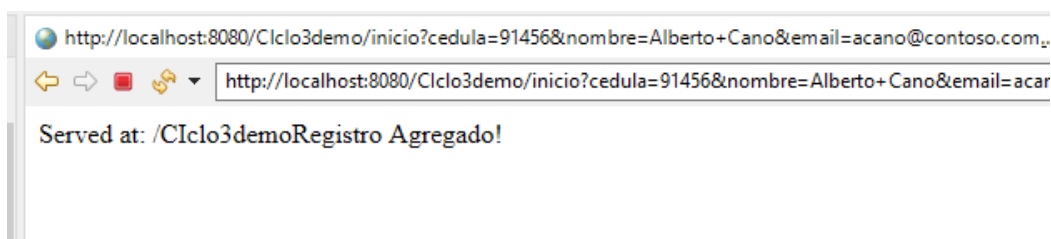


Semana 4

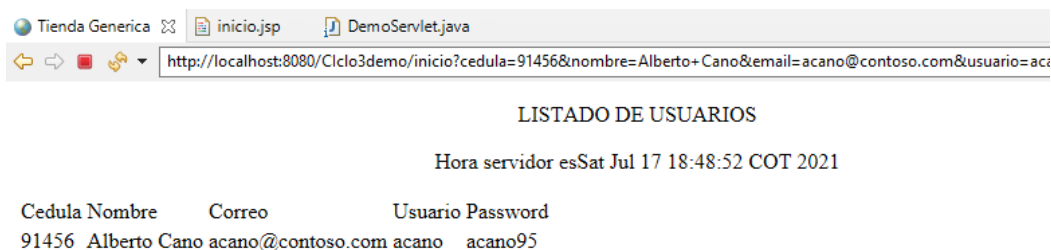
Arquitectura de software parte 2 y metodología
de desarrollo Scrum parte 2.

| | |
|--|--|
| Cedula: | <input type="text" value="91456"/> |
| Nombre: | <input type="text" value="Alberto Cano"/> |
| Correo Electronico: | <input type="text" value="acano@contoso.com"/> |
| Usuario: | <input type="text" value="acano"/> |
| Password: | <input type="password" value="acano95"/> |
| <input type="button" value="Agregar"/> | |
| <input type="button" value="Listar Usuarios"/> | |

Obtendremos el siguiente resultado:



Nos indica que insertamos el registro en la tabla de forma correcta (lo cual su puede verificar, si se desea, con la aplicación MySQL WorkBench). Procedemos a regresar con el botón “Atrás”, y hacemos la consulta, presionando el botón “listar usuarios”.



Con esto, podemos comprobar que nuestra integración entre las aplicaciones de frontend y backend ha sido exitosa.