**Instituto Politécnico Nacional.**

**Escuela Superior De Cómputo.**





**Materia:**

**Desarrollo de Sistemas Distribuidos.**

**Tema:**

**Desarrollo de un cliente para un servicio Web REST.**

**(Tarea 08)**

**Profesor:**

**Carlos Pineda Guerrero.**

**Alumno:**

**Mario Alberto Miranda Sandoval.**

**Grupo:**

**4CM5**

**Objetivo.**

Desarrollar un programa Java consola que consuma el servicio web creado en la tarea 7.

**Desarrollo.**

Para esta tarea al ejecutarse el programa en modo consola, se debe desplegar el siguiente menú:

* Alta usuario.
* Consulta usuario.
* Borra usuario.
* Borra todos los usuarios.
* Salir.

Con la siguiente funcionalidad:

* Alta usuario: Leerá del teclado el email, el nombre de usuario, el apellido paterno, el apellido materno, la fecha de nacimiento, el teléfono y el género (“M” o “F”).
* Consulta usuario: Leerá del teclado el email de un usuario previamente dado de alta.
* Borra usuario: Leerá del teclado el email de un usuario previamente dado de alta.
* Borra todos los usuarios: Borrara todos los usuarios dados de alta previamente.
* Salir: Termina el programa.

Primeramente, creamos tres clases que serán las que nos ayudaran para el control, envío, recepción y despliegue de la información, estas clases son **Usuario.java, Error.java, ResponseModel.java**

**Usuario.java**

1. **public** **class** Usuario {
2. **public** Usuario() {
3. **this**.foto = **null**;
4. }
6. String getEmail() { **return** **this**.email; }
7. String getNombre() { **return** **this**.nombre; }
8. String getApellidoPaterno() { **return** **this**.apellido\_paterno; }
9. String getApellidoMaterno() { **return** **this**.apellido\_materno; }
10. String getFechaNacimiento() { **return** **this**.fecha\_nacimiento; }
11. String getTelefono() { **return** **this**.telefono; }
12. String getGenero() { **return** **this**.genero; }
13. **byte**[] getFoto() { **return** **this**.foto; }
15. **void** setEmail(String email) { **this**.email = email; }
16. **void** setNombre(String nombre) { **this**.nombre = nombre; }
17. **void** setApellidoPaterno(String apellidoPaterno) { **this**.apellido\_paterno = apellidoPaterno; }
18. **void** setApellidoMaterno(String apellidoMaterno) { **this**.apellido\_materno = apellidoMaterno; }
19. **void** setFechaNacimiento(String fechaNacimiento) { **this**.fecha\_nacimiento = fechaNacimiento; }
20. **void** setTelefono(String telefono) { **this**.telefono = telefono; }
21. **void** setGenero(String genero) { **this**.genero = genero; }
22. **void** setFoto(**byte**[] foto) { **this**.foto = foto; }
24. **public** String toString() {
25. **return** "Email: " + email + "\n" +
26. "Nombre: " + nombre + "\n" +
27. "Apellido Paterno: " + apellido\_paterno + "\n" +
28. "Apellido Materno: " + apellido\_materno + "\n" +
29. "Fecha de nacimiento: " + fecha\_nacimiento + "\n" +
30. "Telefono: " + telefono + "\n" +
31. "Genero: " + genero + "\n" +
32. "Foto: null";
33. }
35. **private** String email;
36. **private** String nombre;
37. **private** String apellido\_paterno;
38. **private** String apellido\_materno;
39. **private** String fecha\_nacimiento;
40. **private** String telefono;
41. **private** String genero;
42. **private** **byte**[] foto;
43. }

La clase Usuario.java modela el usuario que el servidor espera al ser invocado el método “alta”.

**Error.java**

1. **public** **class** Error {
2. String message;
4. **public** Error(String message) {
5. **this**.message = message;
6. }
7. }

La clase Error.java modela la recepción del mensaje de error del servidor.

**ResponseModel.java**

1. **public** **class** ResponseModel {
2. **public** ResponseModel(**int** responseCode, String message) {
3. **this**.responseCode = responseCode;
4. **this**.message = message;
5. }
7. **public** **int** getResponseCode() { **return** **this**.responseCode; }
8. **public** String getMessage() { **return** **this**.message; }
10. **public** **void** setResponseCode(**int** responseCode) {
11. **this**.responseCode = responseCode;
12. }
14. **public** **void** setMessage(String message) {
15. **this**.message = message;
16. }
18. **private** **int** responseCode;
19. **private** String message;
20. }

La clase ResponseModel.java sirve como un modelo para recibir el código de respuesta del servidor y el mensaje que este regreso, así se puede diferenciar para mostrar el error en la consola.

Ahora, procederé a explicar la clase **Consumer.java**, esta clase es la encargada de mostrar el menú e invocar los métodos del servicio Web.

1. **protected** **char** mostrarMenu() {
2. Scanner s = **new** Scanner(System.in);
4. System.out.println("a. Alta usuario");
5. System.out.println("b. Consulta usuario");
6. System.out.println("c. Borra usuario");
7. System.out.println("d. Borra todos los usuarios");
8. System.out.println("e. Salir");
10. **char** seleccion = s.nextLine().charAt(0);
12. **return** seleccion;
13. }

El método mostrar menú despliega el menú en consola y lee la opción introducida por el usuario.

1. **protected** **void** opcion(**char** op) {
2. **switch**(op) {
3. **case** 'a':
4. altaUsuario();
5. **break**;
6. **case** 'b':
7. consultaUsuario();
8. **break**;
9. **case** 'c':
10. borrarUsuario();
11. **break**;
12. **case** 'd':
13. borrarTodos();
14. **break**;
15. **case** 'e':
16. System.exit(0);
17. **break**;
18. **default**:
19. System.out.println("No hay mas opciones");
20. **break**;
21. }
22. }

El método opción, se encarga de llamar a los métodos que prepararán la invocación a los servicios.

Antes de pasar a explicar como funciona cada método, se definieron dos constantes con las que se especificara que tipo de servicio se requiere si es GET o POST.

1. **private** **final** String POST\_METHOD = "POST";
2. **private** **final** String GET\_METHOD = "GET";

Estas constantes se explicará su uso más adelante.

1. **private** **void** altaUsuario() {
2. GsonBuilder builder = **new** GsonBuilder();
3. builder.serializeNulls();
5. Gson gson = builder.create();
7. **try** {
8. Usuario usuario = **new** Usuario();
9. usuario = UsuarioUtils.crearUsuario(usuario);
10. String cuerpo = gson.toJson(usuario);
11. ResponseModel response = GenericServices.hacerConsulta(cuerpo, POST\_METHOD, “alta”, “usuario”);
12. **if**(response.getResponseCode() != 400) {
13. System.out.println(response.getMessage());
14. } **else** {
15. Error error = gson.fromJson(response.getMessage(), Error.**class**);
16. System.out.println(error.message);
17. }
18. } **catch**(Exception e) { e.printStackTrace(); }
19. }

El método altaUsuario primero crea un builder de la clase GsonBuilder a modo de poder serializar campos con un valor de null (el atributo de la foto es null), después creamos nuestro objeto gson, como en la clase UsuarioUtils los métodos estáticos pueden arrojar una excepción debido a eso usamos un bloque try-catch, una vez creado el usuario lo serializamos usando el objeto gson, hacemos la llamada para hacer la invocación (más adelante se explicará el funcionamiento de la clase GenericServices), ahora evaluamos la respuesta, (como se sabe que el servidor manda el código 400 cuando hay error, por eso se valida con ese código), si él código de respuesta es exitoso, se muestra la respuesta del servidor, si no lo es, se muestra el mensaje de error.

1. **private** **void** consultaUsuario() {
2. Gson gson = **new** Gson();
4. **try** {
5. String cuerpo = UsuarioUtils.leerEmail();
6. ResponseModel response = GenericServices.hacerConsulta(cuerpo, GET\_METHOD, "consulta", "email");
8. **if**(response.getResponseCode() != 400) {
9. Usuario usuario = gson.fromJson(response.getMessage(), Usuario.**class**);
10. System.out.println(usuario.toString());
11. } **else** {
12. Error error = gson.fromJson(response.getMessage(), Error.**class**);
13. System.out.println(error.message);
14. }
15. } **catch**(Exception e) { e.printStackTrace(); }
16. }
18. **private** **void** borrarUsuario() {
19. Gson gson = **new** Gson();
21. **try** {
22. String cuerpo = UsuarioUtils.leerEmail();
23. ResponseModel response = GenericServices.hacerConsulta(cuerpo, POST\_METHOD, "borra", "email");
25. **if**(response.getResponseCode() != 400) {
26. System.out.println(response.getMessage());
27. } **else** {
28. Error error = gson.fromJson(response.getMessage(), Error.**class**);
29. System.out.println(error.message);
30. }
31. } **catch**(Exception e) { e.printStackTrace(); }
32. }

El método consultaUsuario y borrarUsuario funcionan de manera similar, debido a que ambos piden el email a consultar/borar y hacen la llamada a la invocación del método, la diferencia radica en la invocación al endpoint y el tipo de método (GET o POST), posteriormente se valida y se muestra el mensaje.

1. **private** **void** borrarTodos() {
2. Gson gson = **new** Gson();
4. **try** {
5. ResponseModel response = GenericServices.hacerConsulta("", POST\_METHOD, "borrar", "");
6. **if**(response.getResponseCode() != 400) {
7. System.out.println(response.getMessage());
8. } **else** {
9. Error error = gson.fromJson(response.getMessage(), Error.**class**);
10. System.out.println(error.message);
11. }
12. } **catch**(Exception e) { e.printStackTrace(); }
13. }

El método borrar todos, hace de manera directa la invocación, se puede observar que no tiene un cuerpo como parámetros, ni el nombre del parámetro, posteriormente solo se valida la respuesta del servidor.

**UsuarioUtils.java**

1. **import** java.io.\*;
3. **public** **class** UsuarioUtils {
4. **protected** **static** Usuario crearUsuario(Usuario usuario) **throws** IOException {
5. BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.in));
7. System.out.println("Email:");
8. usuario.setEmail(br.readLine());
10. System.out.println("Nombre:");
11. usuario.setNombre(br.readLine());
13. System.out.println("Apellido Paterno:");
14. usuario.setApellidoPaterno(br.readLine());
16. System.out.println("Apellido Materno:");
17. usuario.setApellidoMaterno(br.readLine());
19. System.out.println("Fecha de nacimiento:");
20. usuario.setFechaNacimiento(br.readLine());
22. System.out.println("Telefono:");
23. usuario.setTelefono(br.readLine());
25. System.out.println("Genero:");
26. usuario.setGenero(br.readLine());
28. **return** usuario;
29. }
31. **protected** **static** String leerEmail() **throws** IOException {
32. String email;
33. BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.in));
35. System.out.println("Introduce Email a consultar");
36. email = br.readLine();
38. **return** email;
39. }
40. }

La clase UsuarioUtils solo contiene dos métodos estáticos, donde el primero es la creación del usuario, mientras que el segundo es la obtención del email a ser consultado.

**GenericServices.java**

1. **import** java.io.\*;
2. **import** java.net.\*;
4. **public** **class** GenericServices {
5. **public** GenericServices() {}
7. **static** ResponseModel hacerConsulta(String cuerpo, String metodo, String endpoint, String parametro) {
8. **try** {
9. URL url = **new** URL(URL\_MAQUINA + endpoint);
10. HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
12. conexion.setDoOutput(**true**);
13. conexion.setRequestMethod(metodo);
14. conexion.setRequestProperty(REQUEST\_KEY, VALUE\_KEY);
16. **if**(cuerpo.length() > 0 && parametro.length() > 0) {
17. String parametros = parametro + "=" + URLEncoder.encode(cuerpo, "UTF-8");
18. OutputStream os = conexion.getOutputStream();
19. os.write(parametros.getBytes(), 0, parametros.getBytes().length);
20. os.flush();
21. os.close();
22. }
24. **if**(conexion.getResponseCode() != HttpURLConnection.HTTP\_OK)
25. **return** **new** ResponseModel(400, "{message: 'No encontrado'}");
27. BufferedReader br = **new** BufferedReader(**new** InputStreamReader(conexion.getInputStream()));
28. String respuestaServidor;
29. String respuesta = "";
30. **while**((respuestaServidor = br.readLine()) != **null**) respuesta += respuestaServidor;
31. conexion.disconnect();
33. **return** **new** ResponseModel(conexion.getResponseCode(), respuesta);
34. } **catch**(Exception e) { e.printStackTrace(); }
36. **return** **new** ResponseModel(404, "No encontrado");
37. }
39. **private** **final** **static** String URL\_MAQUINA = "http://40.124.32.135:8080/Servicio/rest/ws/";
40. **private** **final** **static** String REQUEST\_KEY = "Content-Type";
41. **private** **final** **static** String VALUE\_KEY = "application/x-www-form-urlencoded";
42. }

La clase GenericServices, será la clase que se encargará de hacer la invocación de los métodos, esta clase contiene un método estático (hacerConsulta), que retorna un ResponseModel, este método recibe cuatro parámetros.

* cuerpo: Es un parámetro de tipo String donde se contiene la información a mandar al servicio, puede ser un Gson serializado o solo el email.
* metodo: Es un parámetro de tipo String que recibe el tipo de método a ser invocado, ya sea GET o POST.
* endpoint: Parámetro de tipo String que recibe a que endpoint del servicio llamar.
* parametro: Es de tipo de String, será el que se mande como el identificador del formulario, puede ser email o usuario.

Ahora, tenemos tres constantes, las cuales son el tipo de datos que se mandaran al servidor, y la otra es la url de la máquina, que tiene la IP pública de la máquina de Azure y el puerto 8080.

El método primero establece la conexión con el servicio, de ahí se checa si se mando un cuerpo o un parámetro para el url, de no ser así se sabe que es la llamada al método de borrar todos, si no lo es, se codifican el parámetro de la url con el cuerpo mandado, se abre un flujo de salida y se mandan los bytes, posteriormente se valida la respuesta del servidor y se crea el ResponseModel.

1. **public** **class** Principal {
2. **public** **static** **void** main(String[] args) {
3. Consumer consumer = **new** Consumer();
5. **while**(**true**) {
6. **char** e = consumer.mostrarMenu();
7. consumer.opcion(e);
8. }
9. }
10. }

Por último, en la clase Principal, creamos nuestro consumidor y mandamos a llamar sus métodos.

Ahora para tener el desarrollo completo de esta práctica el servicio también debió ser modificado, el servicio fue modificado en sus métodos del alta\_usuario y la adición del método de borrar todos.

1. **public** Response alta\_usuario(@FormParam("usuario") String user) **throws** Exception
2. {
3. Usuario usuario = j.fromJson(user, Usuario.class);

El método alta\_usuario se modificó, a modo de recibir un String, ya que nosotros mandamos un String serializado con Gson, posteriormente creamos un usuario con el método fromJson, y el método sigue su flujo con normalidad.

1. @POST
2. @Path("borrar")
3. @Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)
4. @Produces(MediaType.APPLICATION\_JSON)
5. **public** Response borrar\_todo() **throws** Exception
6. {
7. Connection conexion= pool.getConnection();
9. **try**
10. {
11. PreparedStatement stmt\_2 = conexion.prepareStatement("DELETE FROM fotos\_usuarios");
12. **try**
13. {
14. stmt\_2.executeUpdate();
15. }
16. **finally**
17. {
18. stmt\_2.close();
19. }
21. PreparedStatement stmt\_3 = conexion.prepareStatement("DELETE FROM usuarios");
22. **try**
23. {
24. stmt\_3.executeUpdate();
25. }
26. **finally**
27. {
28. stmt\_3.close();
29. }
30. }
31. **catch** (Exception e)
32. {
33. **return** Response.status(400).entity(j.toJson(**new** Error(e.getMessage()))).build();
34. }
35. **finally**
36. {
37. conexion.close();
38. }
39. **return** Response.ok().build();
40. }

Ahora el método borrar\_todo, funciona similar al método borra\_usuario, a diferencia que se quitan las consultas de verificar el email, y se procede directamente a eliminar los registros de la base de datos tanto como las fotos y los usuarios.

Ahora, pasaremos a las pruebas, omitiré la configuración de Tomcat ya que esta fue reportada en la Tarea 7 y el procedimiento que se uso fue exactamente el mismo.

Una captura de pantalla de una computadora

Descripción generada automáticamente

*Antes de hacer las pruebas checamos que el puerto 8080 este habilitado.*

Captura de pantalla de computadora

Descripción generada automáticamente

*Ponemos en ejecución nuestro servicio de Tomcat.*

Texto

Descripción generada automáticamente

*Compilamos el programa y lo ejecutamos, vemos como se muestra el menú de opciones.*

Texto

Descripción generada automáticamente

*En la imagen se observa como se da de alta un usuario.*

Texto

Descripción generada automáticamente

*Ahora consultamos el usuario previamente dado de alta y vemos que nos despliega su información.*

Texto

Descripción generada automáticamente

*Dando de alta un segundo usuario.*

Texto

Descripción generada automáticamente

*Dando de alta un tercer usuario.*

Texto

Descripción generada automáticamente

*Ahora salimos, para probar que toda la información está en el servicio.*

Texto

Descripción generada automáticamente

*Volvemos a ejecutar el programa y consultamos para el usuario test3, vemos que nos despliega la información del usuario.*

Texto

Descripción generada automáticamente

*Ahora borramos el usuario test3.*

Texto

Descripción generada automáticamente

*Intentamos buscar el usuario test3 que hemos eliminado y nos manda que no ha sido encontrado.*

Texto

Descripción generada automáticamente

*Ahora borramos todos los usuarios de la base de datos.*

*Recordemos que teníamos 3 usuarios, borramos 1, así que nos quedan 2.*

Texto

Descripción generada automáticamente

*Checamos los usuarios que teníamos aun y vemos que no hay ninguno, dando como resultado que la función funciono correctamente.*

**Conclusión.**

En esta tarea se puede observar cómo funciona un consumidor de un servicio web en java, lo cierto es que un método como el hacerConsulta de la clase GenericServices puede ser mejorado en demasía, separando los tipos de métodos que se usaran, también teniendo la validación de la respuesta del servidor, en los métodos que llaman al hacerConsulta en vez de validarse dentro de esta, a su vez el Gson (al final de todo JSON) es una herramienta de alto valor cuando se hace la comunicación entre backend y frontend.