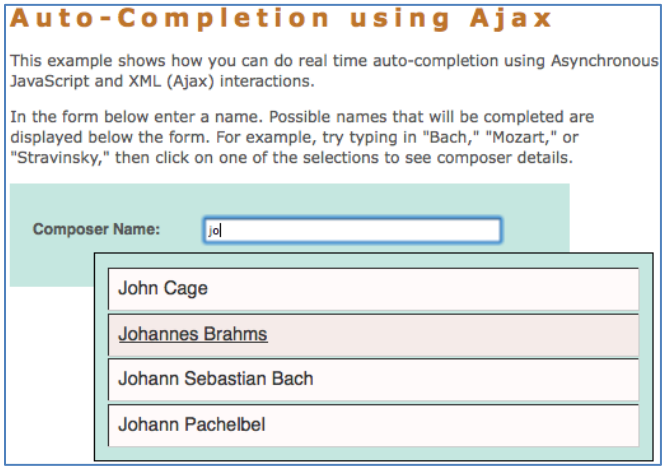
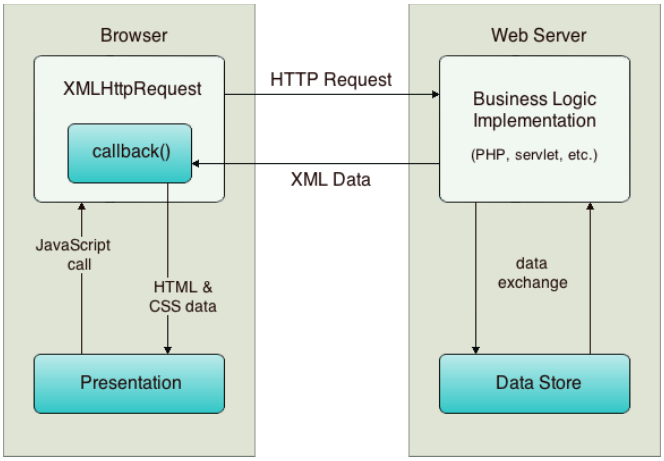
Ajax con Java

Introducción.

En este ejercicio se construye una aplicación con Ajax que emplea el auto-completado en un campo de texto.
Ajax significa Asynchronous JavaScript y XML. Ajax es la forma eficiente de una aplicación web para gestionar las interacciones del usuario con una página web, de una manera que reduce la necesidad de hacer una actualización de la página o página de recarga completa para cada interacción del usuario. Las interacciones Ajax se manejan de forma asíncrona; mientras esto sucede, el usuario puede continuar trabajando con la página. Las interacciones Ajax son iniciadas por el código JavaScript. Cuando la interacción de Ajax se completa, JavaScript actualiza el código fuente HTML de la página. Los cambios se realizan de inmediato sin necesidad de una actualización de la página. Las interacciones Ajax se utilizan para la validación de las entradas del formulario (mientras el usuario las utiliza) utilizando la lógica del lado del servidor, recuperar datos detallados del servidor, actualizar de forma dinámica datos en una página, y presentar datos parciales de la página.
En este caso, se tiene una página web en la que un usuario puede buscar información sobre compositores musicales. La página incluye un campo con una función de autocompletado, donde el usuario introduce el nombre del compositor. El usuario puede escribir parte del nombre del compositor, y la aplicación web intenta completar el nombre haciendo una lista de todos los compositores cuyo nombre, o apellido, comienza con los caracteres introducidos. La función de autocompletado ahorra al usuario el tener que recordar el nombre completo del compositor y proporciona un camino más intuitivo y directo a la información buscada.



La implantación del auto-completado en un campo de búsqueda es algo que se puede realizar con Ajax. Ajax funciona mediante el empleo de un objeto `XMLHttpRequest` para aprobar las solicitudes y respuestas de forma asíncrona entre el cliente y el servidor. El siguiente diagrama ilustra el flujo del proceso de la comunicación que se produce entre el cliente y el servidor.



El flujo del proceso del diagrama se describe en las siguientes etapas:

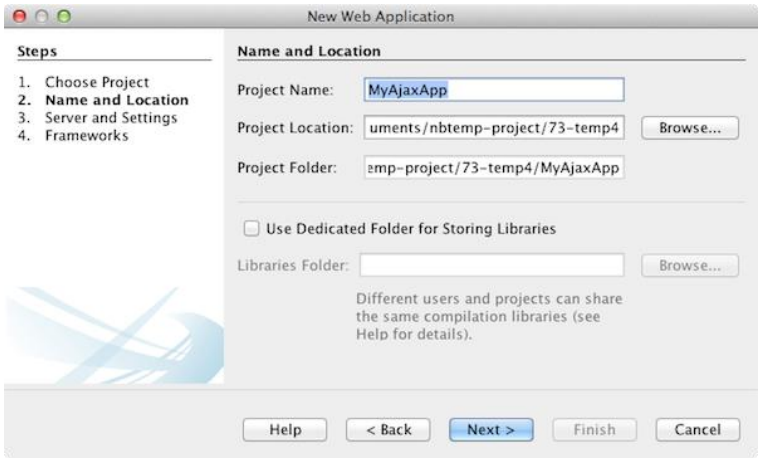
1. El usuario activa un evento, por ejemplo mediante la liberación de una tecla al escribir un nombre. Esto se traduce en una llamada a una función JavaScript que inicializa un objeto `XMLHttpRequest`.
2. El objeto `XMLHttpRequest` está configurado con un parámetro de solicitud que incluye el `ID` del componente que activa el evento, y cualquier valor que el usuario ha introducido. Luego, el objeto `XMLHttpRequest` hace una solicitud asincrónica al servidor web.
3. En el servidor web, un objeto, como un servlet o el escucha se encarga de la solicitud. Los datos se recuperan desde el almacén de datos, y se prepara una respuesta conteniendo los datos en forma de documento XML.
4. Por último, el objeto `XMLHttpRequest` recibe los datos XML utilizando una función para la devolución de llamada, la procesa y actualiza el DOM HTML (Document Object Model) para mostrar la página que contiene los nuevos datos.

Este ejercicio muestra cómo construir el escenario de autocompletar siguiendo el flujo del proceso se indica en el diagrama anterior. En primer lugar, se crean los archivos de cliente para la presentación y funcionalidad necesaria para generar el objeto `XMLHttpRequest`. Luego, configurar el lado del servidor mediante la creación del almacén de datos y la lógica de negocio utilizando tecnología basada en Java. Finalmente, regresar al lado del cliente e implantar `callback`(), y otra funcionalidad JavaScript para actualizar el DOM HTML.

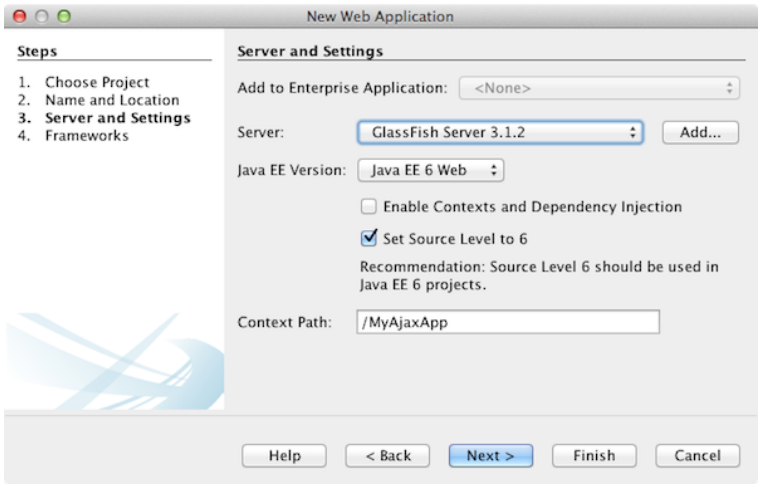
1. Programando del Lado del Cliente: Parte 1

Se inicia con la creación de un nuevo proyecto de aplicación web en el IDE. El IDE contiene plantillas integradas para varios tipos de proyectos.

1. Seleccionar File > New Project. En Categorías, seleccionar `Java Web`. En Projects, seleccionar `Web Application`, clic en Next.
2. En el panel Name and Location, introducir `MyAjaxApp` en Project Name. El campo Project Location especifica la ubicación del proyecto en el equipo. Dejar sin cambios el resto de las opciones con sus valores predeterminados y clic en Next.

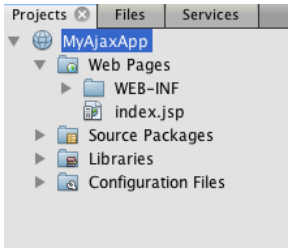


3. En el panel Server and Settings, seleccionar el servidor en el que se desea implantar la aplicación. Sólo se enumeran los servidores que están registrados con el IDE.



4. Aceptar las otras configuraciones predeterminadas y clic en Finish. El proyecto se crea en el sistema de archivos y se abre en el IDE.

Cuando se crean proyectos web basados en Java, se genera automáticamente un script de construcción Ant que permite compilar el proyecto para que se pueda implantar de inmediato y ejecutar en un servidor registrado en el IDE.
Una página predeterminada de entrada se genera y se abre en editor del IDE. Dependiendo del servidor destino, la página de entrada será index.jsp o index.html.



Antes de iniciar la codificación, probar la ejecución de la aplicación para asegurar que la configuración entre el IDE, el servidor y el navegador está configurada correctamente.

1. En la ventana Projects, clic derecho en el nodo del proyecto y seleccionar Run.

La aplicación se compila, se inicia el servidor, despliega la aplicación y la ejecuta. El IDE abre el navegador predeterminado y muestra la página predeterminada de entrada.

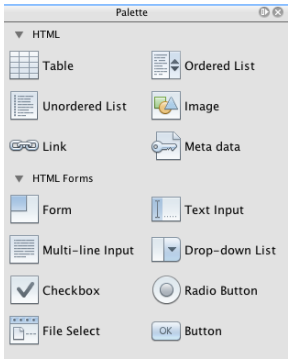
1.1 Uso del editor de HTML

Ahora que el entorno está configurado correctamente, se comienza por transformar la página de índice en la interfaz de auto-completado que será visible para los usuarios.

Una de las ventajas de usar un IDE es que el editor proporciona el completado de código que, si se aprovecha se puede aumentar rápidamente la productividad. El editor del IDE se adapta a la tecnología que utiliza, por lo que si se está trabaja en una página HTML, al digitar la combinación de teclas de completado de código (Ctrl+Espacio) producirá sugerencias de etiquetas y atributos HTML. Lo mismo se aplica para otras tecnologías, como CSS y JavaScript.

Una segunda característica es el uso de la Palette del IDE. La paleta posee plantillas de elementos que se aplican en la tecnología que se está codificando. Simplemente hacer clic en un elemento y arrastrarlo a una ubicación en el archivo del editor de código.

Nota: Se pueden mostrar iconos grandes con clic derecho en la paleta y seleccionar Show Big Icons.



1. Reemplazar el contenido de la etiquetas `<title>` y `<h1>` para leer: `Auto-Completion using AJAX`. La página índice no requiere código script del lado del servidor, por lo que se pueden eliminar remanentes que se crearon de forma predeterminada. La página de índice se debe mostrar como se indica a continuación.

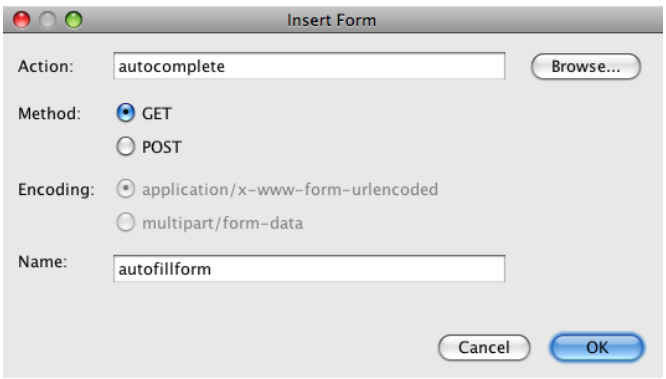
```
<!DOCTYPE html>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Auto-Completion using AJAX</title>
    </head>
    <body>
        <h1>Auto-Completion using AJAX</h1>
    </body>
</html>
```

2. Se agrega texto descriptivo con el propósito del campo de texto. Se puede copiar y pegar en el siguiente texto en un punto justo debajo de las etiquetas `<h1>`:

```
<p>This example shows how you can do real time auto-completion using Asynchronous
    JavaScript and XML (Ajax) interactions.</p>

<p>In the form below enter a name. Possible names that will be completed are displayed
    below the form. For example, try typing in "Bach," "Mozart," or "Stravinsky,"
    then click on one of the selections to see composer details.</p>
```

3. Agregar un formulario HTML a la página. Para hacerlo se hace uso de los elementos listados en la paleta del IDE. Si la paleta no está abierta, seleccionar Window > Palette en el menú principal. Luego, en HTML Forms, clic en y arrastrar un componente del formulario a la página en un punto por debajo de las etiquetas `<p>` que se acaba de agregar. Se abre el cuadro de diálogo Insert Form. Especificar lo siguiente:

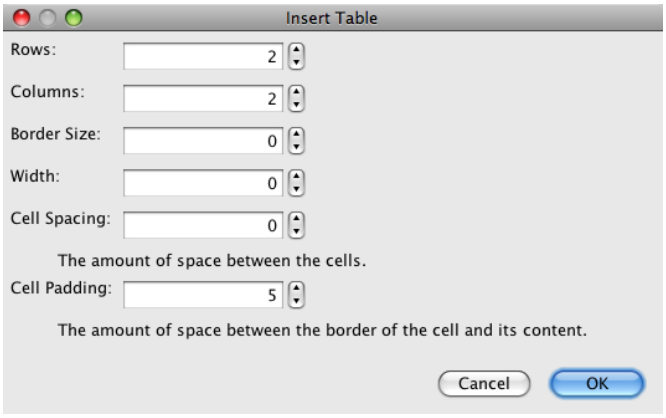
- **Action**: `autocomplete`
- **Méthod** : `GET`
- **Name**: `autofillform`



Clic en OK. Las etiquetas `<form>` de HTML se insertan en la página conteniendo los atributos especificados. (`GET` se aplica por definición, por lo que no se declara explícitamente).

4. Agregar una tabla HTML a la página. En la categoría HTML en la paleta, clic en el componente `Table` y arrastrarlo a un punto entre las etiquetas `<form>`. Se abre el cuadro de diálogo Insert Table. Especificar lo siguiente:

- **Rows** : 2
- **Columns**: 2
- **Border Size**: 0
- **Cell Padding** : 5



5. Clic derecho en el editor y seleccionar Format. Ello formatea el código. El formulario debe mostrarse en forma similar al siguiente:

```
<form name="autofillform" action="autocomplete">
  <table border="0" cellpadding="5">
    <thead>
      <tr>
        <th></th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td></td>
        <td></td>
      </tr>
      <tr>
        <td></td>
        <td></td>
      </tr>
    </tbody>
  </table>
</form>
```

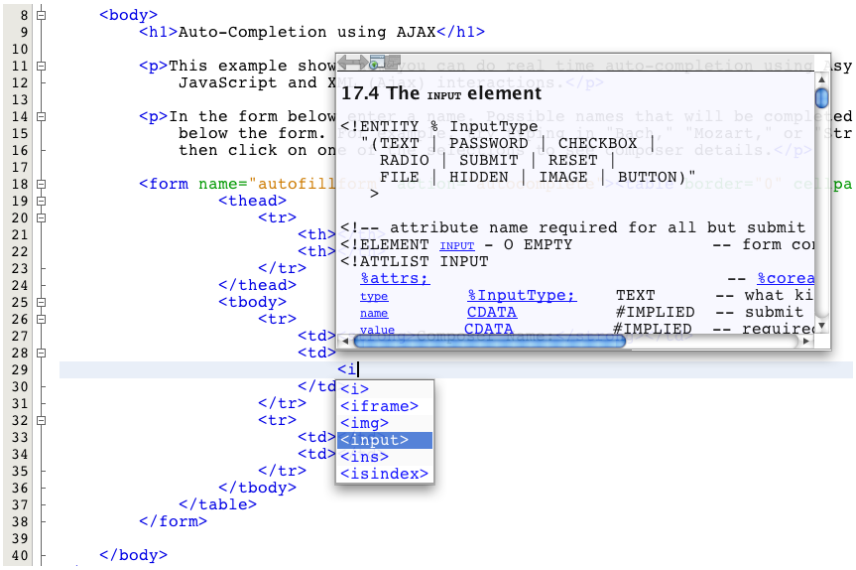
6. En la primera fila de la tabla, ingresar el siguiente texto en la primera columna (cambios en **negritas**):

```
<td><strong>Composer Name:</strong></td>
```

7. En la segunda columna de la primera fila, en lugar de arrastrar un campo Text Input de la paleta , ingresar el siguiente código manualmente (cambios en **negritas**):

```
<td>
    <input type="text"
        size="40"
        id="complete-field"
        onkeyup="doCompletion();">
</td>
```

Cuando se escribe, tratar de usar el completado de código integrado del IDE. Por ejemplo, escribir `<i` , enseguida digitar Ctrl-Space. Se muestra una lista de opciones bajo el cursor, y una descripción del elemento seleccionado se muestra en un cuadro. Se puede digitar Ctrl-Space en cualquier momento de codificación en el editor para que se muestren las posibles opciones. Además, si sólo hay una opción posible, digitando Ctrl-Space se completará automáticamente el nombre del elemento.

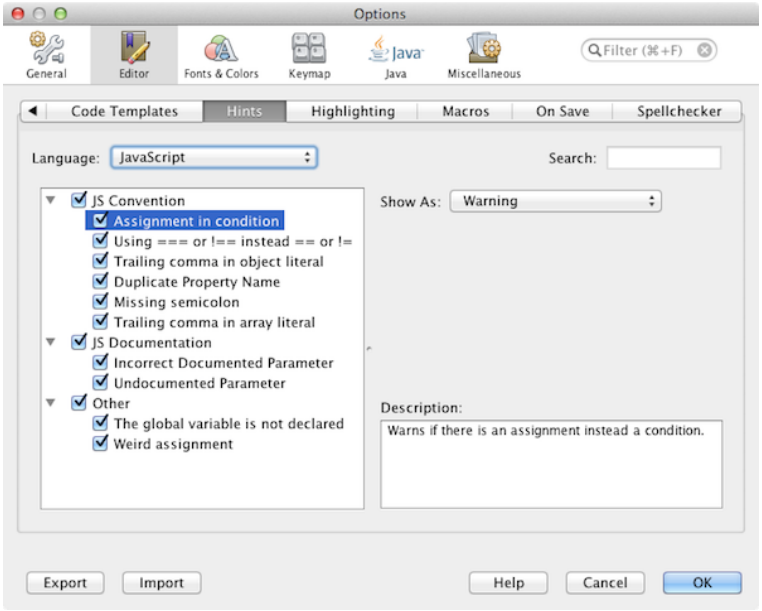


El atributo `onkeyup` que escribió arriba apunta a una función JavaScript llamada `doCompletion`(). Esta función se invoca cada vez que se digita una tecla en el campo de texto del formulario, y se enlaza a la llamada del JavaScript indicado en el diagrama de flujo de Ajax anterior.

1.2 Uso del editor de JavaScript

El editor JavaScript del IDE posee muchas funciones avanzadas de edición, como el completado inteligente de código, resaltado semántico, capacidades inmediatas de renombrado y refactorización, así como otras características.

El completado de código de JavaScript se proporciona automáticamente al código en archivos .js, así como dentro de las etiquetas `<script>` cuando se trabaja con otras tecnologías (por ejemplo, HTML, RHTML, JSP, PHP). El IDE puede proporcionar algunas sugerencias cuando se edita el código JavaScript. Se pueden especificar las opciones de JavaScript seleccionando Tools > Options para abrir la ventana Options y luego seleccionar el lenguaje JavaScript en la pestaña Hints en la categoría Editor. También se pueden añadir plantillas propias de código JavaScript en la pestaña Código Templates en la ventana Options.



Agregar un archivo JavaScript a la aplicación y comenzar la implantación de `doCompletion`().

1. En la ventana Projects, clic en el nodo Web Pages y seleccionar New > JavaScript file. (Si el archivo JavaScript no está en la lista, seleccionar Other. A continuación, seleccionar el archivo JavaScript en la categoría Web en el asistente New File).
2. Nombrar `javascript` al archivo clic en Finish. El nuevo archivo JavaScript se muestra en la ventana Projects en la carpeta Web Pages.
3. Ingresar el siguiente código en `javascript.js`.

```
var req;
var isIE;

function init() {
    completeField = document.getElementById("complete-field");
}

function doCompletion() {
        var url = "autocomplete?action=complete&id=" + escape(completeField.value);
        req = initRequest();
        req.open("GET", url, true);
        req.onreadystatechange = callback;
        req.send(null);
}

function initRequest() {
    if (window.XMLHttpRequest) {
        if (navigator.userAgent.indexOf('MSIE') != -1) {
            isIE = true;
        }
        return new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

Nota: El código anterior realiza una comprobación de compatibilidad del navegador de Firefox 3 (o superior) y las versiones de Internet Explorer (6 y 7, o superior). Si se desea incorporar código más robusto por los problemas de compatibilidad, considerar el uso de un detector de navegadores.

4. Cambiar de nuevo a la página index y agregar una referencia al archivo de JavaScript entre las etiquetas `<head>`.

```
<script type="text/javascript" src="javascript.js"></script>
```

Nota: Se puede alternar rápidamente entre páginas abiertas en el editor digitando Ctrl-Tab.

5. Insertar una llamada a `init`() en la etiqueta <body>.

```
<body onload="init()">
```

Esto asegura que `init`() se llame cada vez que se carga la página.
La tarea de `doCompletion`() es:

- Crear un URL que contiene datos que puede ser utilizados por el lado del servidor,
- inicializar un objeto `XMLHttpRequest`, y
- solicitar al objeto `XMLHttpRequest` que envíe una solicitud asincrónica al servidor.

El objeto `XMLHttpRequest` está en el corazón de Ajax y se ha convertido en el estándar de facto al permitir que los datos XML se pasen de forma asíncrona a través de HTTP. La interacción asíncrona implica que el navegador pueda continuar procesando los eventos en la página después de enviar la solicitud. Los datos se pasan en un segundo plano, y se puede cargar automáticamente en la página sin necesidad de una actualización de la página.

Observar que el objeto `XMLHttpRequest` es creado por `initRequest`(), el cual es invocado por `doCompletion`(). La función comprueba si `XMLHttpRequest` puede ser entendido por el navegador, y si es así se crea un objeto `XMLHttpRequest`. De lo contrario, realiza una comprobación de `ActiveXObject` (el equivalente `XMLHttpRequest` para Internet Explorer 6), y crea un `ActiveXObject` si se le identifica.
Cuando se crea un objeto `XMLHttpRequest` se especifican tres parámetros: una dirección `URL`, el método `HTTP` (`GET` o `POST`), y si la interacción es asíncrona. En el ejemplo anterior, los parámetros son:

- El `autocomplete` de URL y el texto introducido en el `complete-field` por parte del usuario:
  `var url = "autocomplete?action=complete&id=" + escape(completeField.value);`
- `GET` , lo que significa que las interacciones HTTP utilizan el método `GET`, y
- `true` , lo que significa que la interacción es asíncrona :
  `req.open("GET", url, true);`

Si la interacción se establece como asincrónica, se debe especificar una función de devolución de llamada. La función de devolución de llamada de esta interacción se establece con la declaración:
`req.onreadystatechange = callback;`

y la función `callback`( ) se debe definir posteriormente. La interacción HTTP comienza cuando se invoca a `XMLHttpRequest.send`( ). Esta acción se asigna a la solicitud HTTP que se envía al servidor web en el diagrama de flujo anterior.

2. Programando del Lado del Servidor

El IDE proporciona amplio soporte para la programación web en el servidor. Incluye soporte en el editor para muchos lenguajes de programación y script populares, y también soporte a los servicios web, como SOAP, REST, SaaS y frameworks orientados a MVC, como JSF, Spring y Struts. NetBeans Varios plugins están disponibles en el NetBeans Plugin Portal para frameworks con Ajax, incluyendo GWT y Struts2.
La lógica de negocio de la aplicación procesa las solicitudes de recuperación de datos del almacén de datos, y luego la preparación y el envío de la respuesta. Esto se implanta usando un servlet. Antes de comenzar la codificación del servlet, se configura el almacén de datos y la funcionalidad requerida por el servlet de datos de acceso.

2.1 Creación del almacén de datos

Para esta aplicación, se crea una clase llamada `ComposerData` que retiene los datos del compositor utilizando un `HashMap`. Un `HashMap` permite almacenar pares de elementos vinculados en pares clave-valor. También se crea una clase `Composer` que permite al servlet recuperar datos de las entradas en el `HashMap`.

1. Clic derecho en el nodo del proyecto en la ventana de Projects y seleccionar New > Java Class.
2. Nombrar la clase con `ComposerData` e ingresar `com.ajax` en el campo Package. Esto crea un nuevo paquete que contiene la clase, así como otras clases que se crearán posteriormente.
3. Clic en Finish. Se crea la clase y se abre en el editor.
4. En el editor pegar el siguiente código:

```
package com.ajax;

import java.util.HashMap;

/**
 *
 * @author nbuser
 */
public class ComposerData {

    private HashMap composers = new HashMap();

    public HashMap getComposers() {
        return composers;
    }

    public ComposerData() {
```

```
        composers.put("1", new Composer("1", "Johann Sebastian", "Bach", "Baroque"));
        composers.put("2", new Composer("2", "Arcangelo", "Corelli", "Baroque"));
        composers.put("3", new Composer("3", "George Frideric", "Handel", "Baroque"));
        composers.put("4", new Composer("4", "Henry", "Purcell", "Baroque"));
        composers.put("5", new Composer("5", "Jean-Philippe", "Rameau", "Baroque"));
        composers.put("6", new Composer("6", "Domenico", "Scarlatti", "Baroque"));
        composers.put("7", new Composer("7", "Antonio", "Vivaldi", "Baroque"));

        composers.put("8", new Composer("8", "Ludwig van", "Beethoven", "Classical"));
        composers.put("9", new Composer("9", "Johannes", "Brahms", "Classical"));
        composers.put("10", new Composer("10", "Francesco", "Cavalli", "Classical"));
        composers.put("11", new Composer("11", "Fryderyk Franciszek", "Chopin", "Classical"));
        composers.put("12", new Composer("12", "Antonin", "Dvorak", "Classical"));
        composers.put("13", new Composer("13", "Franz Joseph", "Haydn", "Classical"));
        composers.put("14", new Composer("14", "Gustav", "Mahler", "Classical"));
        composers.put("15", new Composer("15", "Wolfgang Amadeus", "Mozart", "Classical"));
        composers.put("16", new Composer("16", "Johann", "Pachelbel", "Classical"));
        composers.put("17", new Composer("17", "Gioachino", "Rossini", "Classical"));
        composers.put("18", new Composer("18", "Dmitry", "Shostakovich", "Classical"));
        composers.put("19", new Composer("19", "Richard", "Wagner", "Classical"));

        composers.put("20", new Composer("20", "Louis-Hector", "Berlioz", "Romantic"));
        composers.put("21", new Composer("21", "Georges", "Bizet", "Romantic"));
        composers.put("22", new Composer("22", "Cesar", "Cui", "Romantic"));
        composers.put("23", new Composer("23", "Claude", "Debussy", "Romantic"));
        composers.put("24", new Composer("24", "Edward", "Elgar", "Romantic"));
        composers.put("25", new Composer("25", "Gabriel", "Faure", "Romantic"));
        composers.put("26", new Composer("26", "Cesar", "Franck", "Romantic"));
        composers.put("27", new Composer("27", "Edvard", "Grieg", "Romantic"));
        composers.put("28", new Composer("28", "Nikolay", "Rimsky-Korsakov", "Romantic"));
        composers.put("29", new Composer("29", "Franz Joseph", "Liszt", "Romantic"));

        composers.put("30", new Composer("30", "Felix", "Mendelssohn", "Romantic"));
        composers.put("31", new Composer("31", "Giacomo", "Puccini", "Romantic"));
        composers.put("32", new Composer("32", "Sergei", "Rachmaninoff", "Romantic"));
        composers.put("33", new Composer("33", "Camille", "Saint-Saens", "Romantic"));
        composers.put("34", new Composer("34", "Franz", "Schubert", "Romantic"));
        composers.put("35", new Composer("35", "Robert", "Schumann", "Romantic"));
        composers.put("36", new Composer("36", "Jean", "Sibelius", "Romantic"));
        composers.put("37", new Composer("37", "Bedrich", "Smetana", "Romantic"));
        composers.put("38", new Composer("38", "Richard", "Strauss", "Romantic"));
        composers.put("39", new Composer("39", "Pyotr Il'yich", "Tchaikovsky", "Romantic"));
        composers.put("40", new Composer("40", "Guiseppe", "Verdi", "Romantic"));

        composers.put("41", new Composer("41", "Bela", "Bartok", "Post-Romantic"));
        composers.put("42", new Composer("42", "Leonard", "Bernstein", "Post-Romantic"));
        composers.put("43", new Composer("43", "Benjamin", "Britten", "Post-Romantic"));
        composers.put("44", new Composer("44", "John", "Cage", "Post-Romantic"));
        composers.put("45", new Composer("45", "Aaron", "Copland", "Post-Romantic"));
        composers.put("46", new Composer("46", "George", "Gershwin", "Post-Romantic"));
        composers.put("47", new Composer("47", "Sergey", "Prokofiev", "Post-Romantic"));
        composers.put("48", new Composer("48", "Maurice", "Ravel", "Post-Romantic"));
        composers.put("49", new Composer("49", "Igor", "Stravinsky", "Post-Romantic"));
        composers.put("50", new Composer("50", "Carl", "Orff", "Post-Romantic"));

    }
}
```

Se observará que se muestra una advertencia en el margen izquierdo del editor porque la clase `Composer` no se puede encontrar. Realizar los siguientes pasos para crear la clase `Composer`.

1. Clic derecho en el nodo del proyecto en la ventana Projects y seleccionar New > Java Class.
2. Nombrar la clase con `Composer` y seleccionar `com.ajax` de la lista desplegable en el campo Package. Clic en Finish. Al hacer clic en Finish el IDE crea la clase y se abre el archivo en el editor.
3. En el editor pegar el siguiente código:

```
package com.ajax;
```

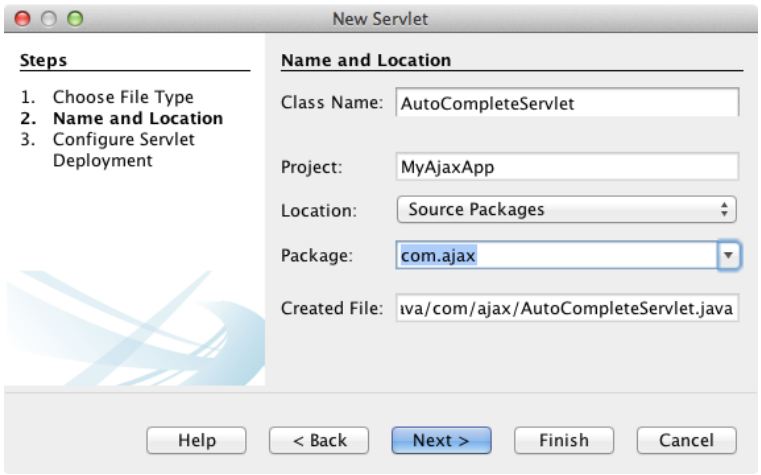
```
public class Composer {

    private String id;
    private String firstName;
    private String lastName;
    private String category;

    public Composer (String id, String firstName, String lastName, String category) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.category = category;
    }

    public String getCategory() {
        return category;
    }

    public String getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

Después de crear la clase `Composer`, observar la clase ComposerData en el editor y ver que las anotaciones de advertencia ya no están ahí. Si se permanecen las anotaciones de advertencia en `ComposerData` se puede tratar de resolver el error añadiendo las declaraciones de importación que faltan.

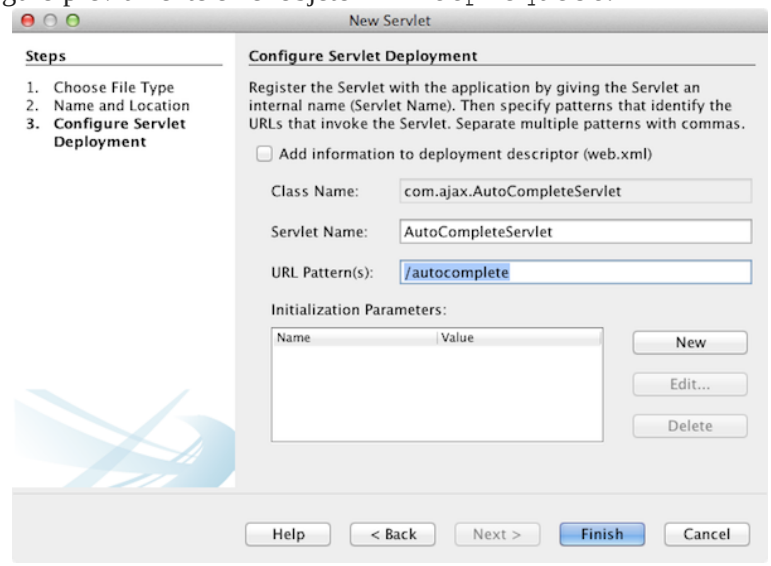
2.2 Creación del Servlet

Se crea un servlet para manejar la URL `autocomplete` que es recibida por la solicitud entrante.

1. Clic derecho en el nodo del proyecto en la ventana de Projects y seleccionar New > Servlet para abrir el asistente New Servlet. (Seleccionar `Other` y seleccionar `Servlet` de la categoría Web si el Servlet no se muestra por definición en el menú emergente).
2. Nombrar el servlet con `AutoCompleteServlet` y seleccionar `com.ajax` en la lista desplegable del campo Package. Clic en Next.



3. En el panel Configure Servlet Deployment, cambiar el patrón URL a `/autocomplete` para que coincida con la URL que se configuró previamente en el objeto `XMLHttpRequest`.



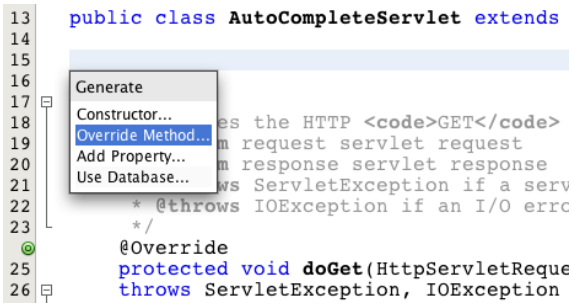
Este panel ahorra el tener que agregar manualmente estos detalles en el descriptor de despliegue.

4. Opcionalmente, seleccionar "Add servlet information to deployment descriptor". Esto es porque el proyecto es el mismo que el descargado de muestra. Con las versiones posteriores del IDE, por definición el servlet se registra con una anotaciónWebServlet y no en un descriptor de despliegue. El proyecto seguirá funcionando si se utiliza la anotación `@WebServlet` en lugar de un descriptor de despliegue.
5. Clic en Finish. Se crea el servlet y se abre en el editor.

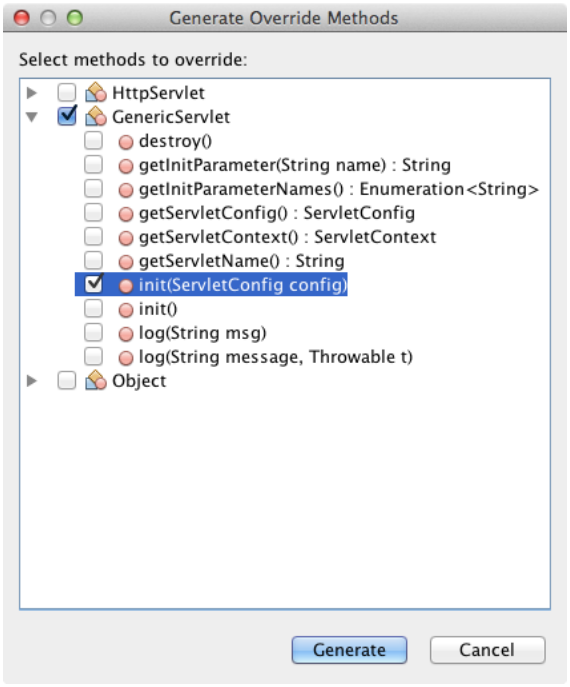
Los únicos métodos que requiere sobreescibir son `doGet`(), para definir cómo el servlet controla la solicitud `autocomplete GET`, e `init`(), que necesita para iniciar un `ServletContext` por lo que el servlet puede acceder a otras clases en la aplicación una vez que se puso en servicio.

Se pueden sobreescribir métodos de las superclases utilizando menú emergente Insert Code del IDE. Implantar `init`() con los pasos siguientes.

1. Colocar el cursor debajo de la declaración de la clase `AutoCompleteServlet` en el editor. Digitar Alt-Insert para abrir el menú emergente Generate Code.



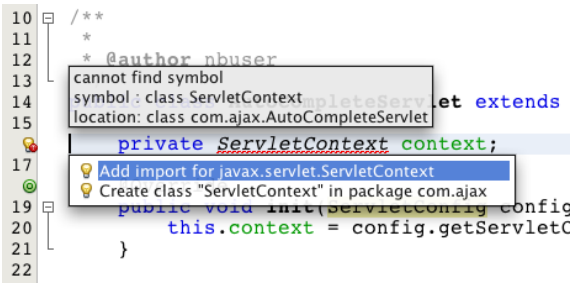
2. Seleccionar Override Method. En el cuadro de diálogo, se muestran todas las clases de las que `AutoCompleteServlet` hereda. Expandir el nodo GenericServlet y seleccionar `init(Servlet Config config)`.



3. Clic en OK. El método `init()` se añade en el editor.
4. Agregar una variable al objeto `ServletContext` y modificar `init()` (cambios en **negritas**):

```
private ServletContext context;

@Override
public void init(ServletConfig config) throws ServletException {
    this.context = config.getServletContext();
}
```

5. Agregar un enunciado `import` para `ServletContext`. Se puede hacer clic en el icono de bombilla que se muestra en el margen izquierdo del editor:



El método `doGet()` tiene que convertir la URL de la solicitud, extraer los datos del almacén de datos, y preparar una respuesta en formato XML. Notar que la declaración del método se genera al crear la clase. Para verlo, se pueden ampliar los métodos `HttpServlet` haciendo clic en el icono de ampliación (⊞) en el margen izquierdo.

1. Agregar las siguientes declaraciones de variables debajo de la declaración de la clase `AutocompleteServlet`.

```
private ComposerData compData = new ComposerData();
private HashMap composers = compData.getComposers();
```

Esto crea un `HashMap` de todos los datos `composer`, que luego lo emplea `doGet()`.

2. Localizar `doGet()` e implantar el método de la siguiente manera:

```
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

    String action = request.getParameter("action");
```

```
    String targetId = request.getParameter("id");
    StringBuffer sb = new StringBuffer();

    if (targetId != null) {
        targetId = targetId.trim().toLowerCase();
    } else {
        context.getRequestDispatcher("/error.jsp").forward(request, response);
    }

    boolean namesAdded = false;
    if (action.equals("complete")) {

        // check if user sent empty string
        if (!targetId.equals("")) {

            Iterator it = composers.keySet().iterator();

            while (it.hasNext()) {
                String id = (String) it.next();
                Composer composer = (Composer) composers.get(id);

                if ( // targetId matches first name
                     composer.getFirstName().toLowerCase().startsWith(targetId) ||
                     // targetId matches last name
                     composer.getLastName().toLowerCase().startsWith(targetId) ||
                     // targetId matches full name
                     composer.getFirstName().toLowerCase().concat(" ")
                        .concat(composer.getLastName().toLowerCase()).startsWith(targetId)) {

                    sb.append("<composer>");
                    sb.append("<id>" + composer.getId() + "</id>");
                    sb.append("<firstName>" + composer.getFirstName() + "</firstName>");
                    sb.append("<lastName>" + composer.getLastName() + "</lastName>");
                    sb.append("</composer>");
                    namesAdded = true;
                }
            }
        }

        if (namesAdded) {
            response.setContentType("text/xml");
            response.setHeader("Cache-Control", "no-cache");
            response.getWriter().write("<composers>" + sb.toString() + "</composers>");
        } else {
            //nothing to show
            response.setStatus(HttpServletResponse.SC_NO_CONTENT);
        }
    }
    if (action.equals("lookup")) {

        // put the target composer in the request scope to display
        if ((targetId != null) && composers.containsKey(targetId.trim())) {
            request.setAttribute("composer", composers.get(targetId));
            context.getRequestDispatcher("/composer.jsp").forward(request, response);
        }
    }
}
```

Como se puede ver en este servlet, no hay nada realmente nuevo que se necesite aprender para escribir código del lado del servidor para el procesamiento de Ajax. El tipo de contenido de la respuesta debe ser establecido a `text/xml` para los casos en los que se desee intercambiar documentos XML. Con Ajax, también puede intercambiar texto plano o incluso fragmentos de JavaScript que pueden ser evaluados o ejecutados por la función de devolución de llamada en el cliente. Notar también que algunos navegadores pueden almacenar en caché los resultados, por lo que puede ser necesario establecer el encabezado `Cache-Control HTTP` con `no-cache`.

En este caso, el servlet genera un documento XML que contiene todos los compositores con un nombre o apellido que comienza con los caracteres escritos por el usuario. Este documento se asigna a los datos XML mostrado en el diagrama de flujo anterior. Enseguida se muestra un ejemplo de un documento XML que se devuelve al objeto `XMLHttpRequest`:

```
<composers>
    <composer>
        <id>12</id>
        <firstName>Antonin</firstName>
        <lastName>Dvorak</lastName>
    </composer>
    <composer>
        <id>45</id>
        <firstName>Aaron</firstName>
        <lastName>Copland</lastName>
    </composer>
    <composer>
        <id>7</id>
        <firstName>Antonio</firstName>
        <lastName>Vivaldi</lastName>
    </composer>
    <composer>
        <id>2</id>
        <firstName>Arcangelo</firstName>
        <lastName>Corelli</lastName>
    </composer>
</composers>
```

Nota: Se puede utilizar Monitor HTTP del IDE para ver los datos XML devueltos una vez que se completa la solicitud.

3. Programando del Lado del Cliente: Parte 2

Ahora se debe definir la función de devolución de llamada para manejar la respuesta del servidor y añadir la funcionalidad necesaria que refleje los cambios en la página que el usuario visualiza. Es necesario modificar el HTML DOM. Es necesario crear páginas JSP para mostrar los resultados de una solicitud correcta o los mensajes de error de una solicitud fallida. Entonces se puede crear una hoja de estilo sencilla de la presentación.

3.1 Adición de funciones para la devolución de la llamada

La función de devolución de llamada se invoca asíncronamente en puntos específicos durante la interacción HTTP cuando cambia la propiedad `readyState` del objeto `XMLHttpRequest`. En la aplicación que está construyendo, la función de devolución de llamada es `callback`(). Se debe recordar que en `doCompletion`(), `callback` se establece como la propiedad `XMLHttpRequest.onreadystatechange` para una función. Ahora, se implanta la función `callback` de la siguiente manera.

1. Abrir Javascript.js en el editor de código fuente e ingresar el código siguiente.

```
function callback() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            parseMessages(req.responseXML);
        }
    }
}
```

Un valor de `readyState` de "4" significa la finalización de la interacción de HTTP. El API de `XMLHttpRequest.readState` indica que hay 5 valores posibles que se pueden establecer. Estos son:

readyState Value	Object Status Definition
0	uninitialized
1	loading
2	loaded
3	interactive

4	complete

Observar que la función `parseMessages`() se invoca sólo cuando el `XMLHttpRequest.readyState` es "`4`" y el estado (la definición código de estado HTTP de la solicitud) es "`200`", lo que significa un éxito. Enseguida de define `parseMessages`() en Actualización del DOM HTML.

3.2 Actualización del DOM HTML

La función `parseMessages`() maneja los datos XML entrantes. Al hacerlo, se basa en varias funciones auxiliares, como `appendComposer`(), `getElementY`(), y `clearTable`(). También se deben introducir nuevos elementos a la página index, como una segunda tabla HTML que sirve de cuadro de autocompletar, e identificadores para los elementos para que puedan ser referenciados en el javascript.js. Por último, se crean nuevas variables que se corresponden con los elementos de la página índice, inicializarlos en la función `init`() que se implantó con anterioridad, y añadir alguna funcionalidad que se necesite cada vez que se cargue la página índice.

Nota: Las funciones y elementos que se creen en los siguientes pasos funcionan de manera interdependiente. Se recomienda que cuando se trabaje en esta sección, se examine el código una vez que esté todo en su lugar.

1. Abrir la página index en el editor y escribir el código siguiente en la segunda fila de la tabla HTML que se creó anteriormente.

```
<tr>
    <td id="auto-row" colspan="2">
        <table id="complete-table" />
    </td>
</tr>
```

La segunda fila de la tabla contiene otra tabla HTML. Esta tabla representa el cuadro de auto-completar que se utiliza para rellenar los nombres del compositor.

2. Abrir el javascript.js en el editor e ingresar las siguientes tres variables en la parte superior del archivo.

```
var completeField;
var completeTable;
var autoRow;
```

3. Agregar las siguientes líneas (en **negritas**) a la función `init`().

```
function init() {
    completeField = document.getElementById("complete-field");
    completeTable = document.getElementById("complete-table");
    autoRow = document.getElementById("auto-row");
    completeTable.style.top = getElementY(autoRow) + "px";
}
```

Uno de los propósitos de `init`() es hacer que los elementos dentro de la página de índice sean accesibles a otras funciones que modificarán el DOM de la página índice.

4. Agregar `appendComposer` () a javascript.js.

```
function appendComposer(firstName,lastName,composerId) {

    var row;
    var cell;
    var linkElement;

    if (isIE) {
        completeTable.style.display = 'block';
        row = completeTable.insertRow(completeTable.rows.length);
        cell = row.insertCell(0);
    } else {
        completeTable.style.display = 'table';
        row = document.createElement("tr");
        cell = document.createElement("td");
        row.appendChild(cell);
```

```
        completeTable.appendChild(row);
    }

    cell.className = "popupCell";

    linkElement = document.createElement("a");
    linkElement.className = "popupItem";
    linkElement.setAttribute("href", "autocomplete?action=lookup&id=" + composerId);
    linkElement.appendChild(document.createTextNode(firstName + " " + lastName));
    cell.appendChild(linkElement);
}
```

Esta función crea una nueva fila de la tabla, le inserta un enlace a un compositor utilizando los datos pasados a la función a través de sus tres parámetros, e inserta la fila en el elemento `complete-table` de la página índice.

5. Agregar `getElementY`() a javascript.js.

```
function getElementY(element){

    var targetTop = 0;

    if (element.offsetParent) {
        while (element.offsetParent) {
            targetTop += element.offsetTop;
            element = element.offsetParent;
        }
    } else if (element.y) {
        targetTop += element.y;
    }
    return targetTop;
}
```

Esta función se usa para encontrar la posición vertical del elemento padre. Esto es necesario debido a que la posición real del elemento, cuando se muestra, a menudo depende del tipo y versión del navegador. Notar que el elemento complete-table, cuando se muestra conteniendo los nombres de compositor, se desplaza a la parte inferior derecha de la tabla en la que existe. El posicionamiento de la altura correcta se determina con `getElementY`().

6. Agregar `clearTable`() a javascript.js.

```
function clearTable() {
    if (completeTable.getElementsByTagName("tr").length > 0) {
        completeTable.style.display = 'none';
        for (loop = completeTable.childNodes.length -1; loop >= 0 ; loop--) {
            completeTable.removeChild(completeTable.childNodes[loop]);
        }
    }
}
```

Esta función establece la visualización del elemento `complete-table` con `'none'`, (es decir, lo hace invisible), y elimina las entradas actuales del nombre del compositor que se crearon.

7. Modificar la función `callback` () para llamar `clearTable` ( ) cada vez que se reciben nuevos datos desde el servidor. Por lo tanto, cualquier entrada de un compositor que ya existe en el cuadro de autocompletado se retiran antes de que se llene con nuevas entradas.

```
function callback() {

    clearTable();

    if (req.readyState == 4) {
        if (req.status == 200) {
            parseMessages(req.responseXML);
        }
```

```
    }
}
```

8. Añadir `parseMessages()` a javascript.js.

```
function parseMessages(responseXML) {

    // no matches returned
    if (responseXML == null) {
        return false;
    } else {

        var composers = responseXML.getElementsByTagName("composers")[0];

        if (composers.childNodes.length > 0) {
            completeTable.setAttribute("bordercolor", "black");
            completeTable.setAttribute("border", "1");

            for (loop = 0; loop < composers.childNodes.length; loop++) {
                var composer = composers.childNodes[loop];
                var firstName = composer.getElementsByTagName("firstName")[0];
                var lastName = composer.getElementsByTagName("lastName")[0];
                var composerId = composer.getElementsByTagName("id")[0];
                appendComposer(firstName.childNodes[0].nodeValue,
                    lastName.childNodes[0].nodeValue,
                    composerId.childNodes[0].nodeValue);
            }
        }
    }
}
```

La función `parseMessages( )` recibe como parámetro una representación de objeto del documento XML devuelto por el servlet `Autocomplete`. La función atraviesa programáticamente el documento XML, extrayendo `firstName`, `lastName` , y el `id` de cada entrada, a continuación, pasa estos datos al `appendComposer()`. Esto resulta en una actualización dinámica de los contenidos del elemento `complete-table`. Por ejemplo, una entrada que se genera y se inserta en `complete-table` puede tener un aspecto como sigue:

```
<tr>
    <td class="popupCell">
        <a class="popupItem" href="autocomplete?action=lookup&id=12">Antonin Dvorak</a>
    </td>
</tr>
```

La actualización dinámica al elemento `complete-table` representa el último paso del flujo del proceso de la comunicación que tiene lugar durante la comunicación utilizando Ajax. Esta actualización se asigna a los datos HTML y CSS que se envían a la presentación en el diagrama de flujo anterior.

Muestra de los Resultados

Para mostrar los resultados, se necesita un archivo JSP llamado `composers.jsp`. Esta página se llama desde `AutoCompleteServlet` durante una acción de consulta. También es necesario un archivo `error.jsp`, que se llama desde `AutoCompleteServlet` si el compositor no se puede encontrar.

Para mostrar los resultados y errores:

1. En la ventana Projects, clic en la carpeta Web Pages de la aplicación y seleccionar New > JSP. Se abre el asistente New JSP.
2. En el campo File Name, ingresar `compositor`. En el campo Created File, se debería ver una ruta que termina en `/web/composer.jsp`.

3. Clic en Finish. El archivo `composer.jsp` se abre en el editor. Un nodo del archivo se mostrará en la ventana Projects en la carpeta Web Pages.
4. Reemplazar el código con el marcador de posición en `composer.jsp` con el siguiente código :

```
<html>
  <head>
    <title>Composer Information</title>

    <link rel="stylesheet" type="text/css" href="stylesheet.css">
  </head>
  <body>

    <table>
      <tr>
        <th colspan="2">Composer Information</th>
      </tr>
      <tr>
        <td>First Name: </td>
        <td>${requestScope.composer.firstName}</td>
      </tr>
      <tr>
        <td>Last Name: </td>
        <td>${requestScope.composer.lastName}</td>
      </tr>
      <tr>
        <td>ID: </td>
        <td>${requestScope.composer.id}</td>
      </tr>
      <tr>
        <td>Category: </td>
        <td>${requestScope.composer.category}</td>
      </tr>
    </table>

    <p>Go back to <a href="index.html" class="link">application home</a>.</p>
  </body>
</html>
```

Nota. Se tendrá que cambiar el enlace para volver a la página índice si la página de índice es `index.jsp`.

5. Crear otro archivo JSP en la carpeta Web Pages del proyecto. Nombrar al archivo con `error.jsp`.
6. Reemplazar el código con el marcador de posición en el `error.jsp` con el siguiente código :

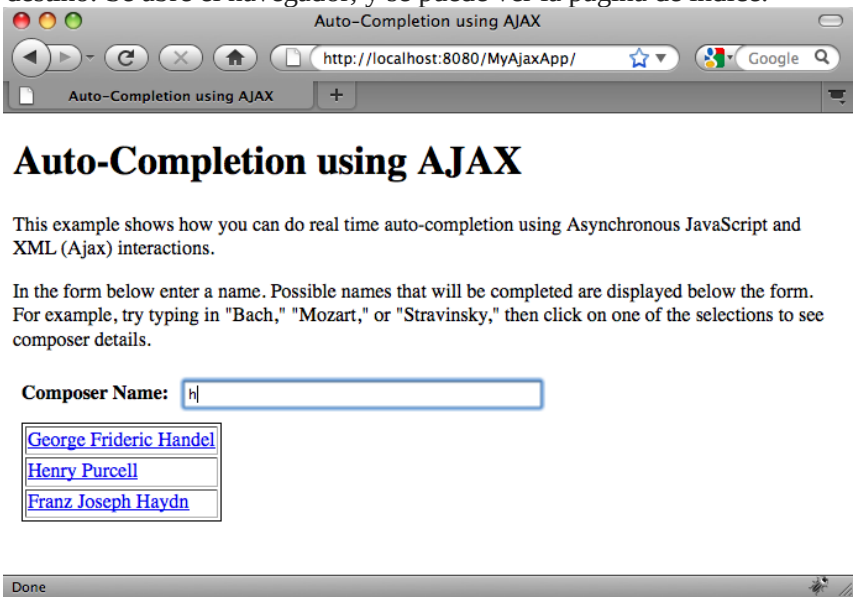
```
<!DOCTYPE html>

<html>
    <head>
        <link rel="stylesheet" type="text/css" href="stylesheet.css">
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Seach Error</title>
    </head>
    <body>
        <h2>Seach Error</h2>

        <p>An error occurred while performing the search. Please try again.</p>

        <p>Go back to <a href="index.html" class="link">application home</a>.</p>
    </body>
</html>
```

Nota. Se tendrá que cambiar el enlace para volver a la página de índice si su página de índice es `index.jsp`.

4. Enlazado de la hoja de estilos

En esta etapa, se ha completado todo el código necesario para la funcionalidad de la aplicación. Para ver los resultados intentar ejecutar la aplicación ahora.

1. En la ventana Projects, clic en el nodo del proyecto y seleccione Run. El proyecto se vuelve a compilar y despliega en el servidor de destino. Se abre el navegador, y se puede ver la página de índice:



Para agregar una hoja de estilo a la aplicación, basta con crear un archivo `.css` y enlazarlo desde la(s) página(s) de presentación. Cuando se trabaja en los archivos `.css`, el IDE proporciona permite el completado de código, así como las siguientes ventanas que ayudan a generar y editar las reglas de las hojas de estilo.

- **Ventana CSS Styles**. La ventana de estilos CSS permite editar las declaraciones de las reglas para los elementos HTML y selectores en un archivo CSS.
- **Cuadro de diálogo Create CSS Rules**. El cuadro de diálogo para crear reglas CSS permite crear nuevas reglas en una hoja de estilos CSS.
- **Cuadro de diálogo Add CSS Property**. El cuadro de diálogo para agregar las propiedades de CSS permite añadir declaraciones a una regla CSS en una hoja de estilo mediante la adición de propiedades y valores.

Para agregar una hoja de estilo a su aplicación se llevan a cabo los siguientes pasos.

1. En la ventana Projects, clic derecho en el nodo Web Pages y seleccionar New > Cascading Style Sheet (Si Cascading Style Sheet no está en la lista, elija `Other`. Luego, seleccionar Cascading Style Sheet de la categoría Web en el asistente New File).
2. En el campo de texto CSS File Name, ingresar **`stylesheet`**. Clic en Finish.

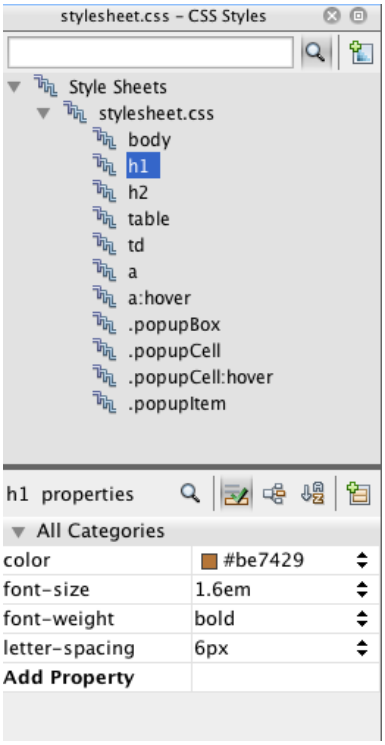
Se crea el nuevo archivo y se abre en el editor.

3. Ingresar las siguientes reglas en `stylesheet.css` en el editor. Se puede hacer uso del completado de código del IDE digitando Ctrl-Space en los puntos cuando se quiere desee invocar las sugerencias.

```
body {
   font-family: Verdana, Arial, sans-serif;
   font-size: smaller;
   padding: 50px;
   color: #555;
   width: 650px;
}

h1 {
   letter-spacing: 6px;
   font-size: 1.6em;
   color: #be7429;
```

```
   font-weight: bold;
}

h2 {
   text-align: left;
   letter-spacing: 6px;
   font-size: 1.4em;
   color: #be7429;
   font-weight: normal;
   width: 450px;
}

table {
   width: 550px;
   padding: 10px;
   background-color: #c5e7e0;
}

td {
   padding: 10px;
}

a {
  color: #be7429;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

.popupBox {
  position: absolute;
  top: 170px;
  left: 140px;
}

.popupCell {
   background-color: #fffafa;
}

.popupCell:hover {
  background-color: #f5ebe9;
}

.popupItem {
  color: #333;
  text-decoration: none;
  font-size: 1.2em;
}
```

4. Abrir la ventana de CSS Styles seleccionando Window > Web > CSS.



Se puede utilizar la ventana de CSS Styles para ver rápidamente las propiedades y reglas de estilo de edición. Al seleccionar una regla en el panel superior de la ventana de estilos CSS se pueden ver las propiedades de la regla en el panel inferior. Se pueden agregar reglas CSS a la hoja de estilo haciendo clic en el icono Edit CSS Rules en la barra de herramientas del panel superior. Se pueden modificar las reglas en el panel inferior editando la hoja de propiedades y añadir propiedades haciendo clic en el icono Add Property en la barra de herramientas del panel inferior.

5. Cambiar a la página índice en el editor, y añadir una referencia a la hoja de estilos entre las etiquetas `<head>`.

```
<link rel="stylesheet" type="text/css" href="stylesheet.css">
```

6. Agregar la clase `popupBox` que está definida en la hoja de estilo al elemento `complete-table` (cambios en **negritas**).

```
<tr>
    <td id="auto-row" colspan="2">
        <table id="complete-table" class="popupBox" />
    </td>
</tr>
```

Se puede utilizar el autocompletado de código en el editor para ayudar a seleccionar la regla de estilo que desea aplicar al selector.

Como se indica en el `stylesheet.css`, esta regla posiciona el elemento `complete-table` para que se muestre un poco a la derecha de su elemento padre.

Al guardar la página de índice de la aplicación se reubicó automáticamente al servidor. Si la página sigue abierta en el navegador se puede recargar la página para ver que la página se encuentra actualizada de acuerdo con las reglas en la hoja de estilos CSS.

5. Ejecución del Proyecto

Al ejecutar de nuevo la aplicación, se muestra en el navegador utilizando la hoja de estilo que se acaba de crear. Cada vez que se ingresa un carácter, una solicitud asíncrona es enviada al servidor, y regresada con datos XML que han sido preparados por `AutoCompleteServlet`. Al ingresar más caracteres, el número de nombres de compositores disminuye para reflejar la nueva lista de coincidencias.

5.1 Uso del monitor HTTP Server

Se puede utilizar el monitor HTTP Server del IDE para verificar la comunicación HTTP que se lleva a cabo mientras las solicitudes y las respuestas se pasan entre el cliente y el servidor. El monitor de HTTP Server muestra información, como los encabezados del cliente y servidor, propiedades de la sesión, los detalles de cookies, así como los parámetros de la solicitud.

Antes de comenzar a utilizar el monitor HTTP, primero se le debe habilitar en el servidor que se está utilizando.

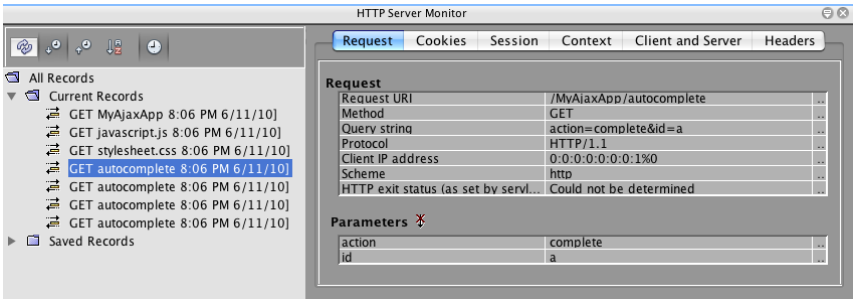
1. Abrir la ventana Servers seleccionando Tools > Servers en el menú principal.
2. En el panel izquierdo, seleccionar el servidor que se está utilizando con el proyecto. A continuación, en el panel derecho, seleccionar la opción Enable HTTP Monitor.

Nota: Esta opción se muestra en la pestaña Common del servidor GlassFish. En Tomcat, reside en la pestaña Connection.

3. Clic en Close.

Nota: Si el servidor ya está en ejecución, es necesario reiniciarlo con el fin de permitir que los cambios surtan efecto. Se puede reiniciar el servidor abriendo la ventana Services (Window>Services), a continuación, clic derecho en el servidor bajo el nodo Servers y seleccionando Restart.

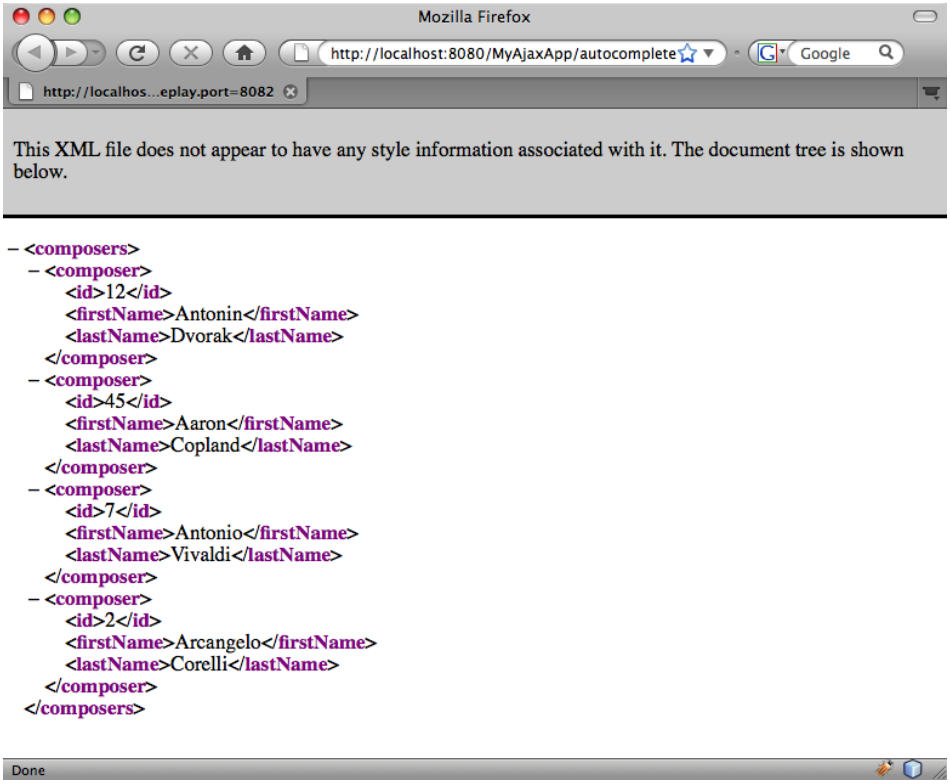
Ahora, al ejecutar de nuevo la aplicación, el monitor HTTP se abre en la región inferior del IDE. Se puede seleccionar un registro en el panel izquierdo, luego clic en las pestañas en la ventana principal para ver la información relativa a cada solicitud que se hace.



Se pueden verificar los datos XML que se envían desde el servidor como resultado de la solicitud asíncrona realizada cuando un usuario ingresa un carácter en el campo de autocompletado.

1. En la vista de árbol en el lado izquierdo del monitor HTTP, clic derecho en un registro de la solicitud y seleccione Replay.

La respuesta se genera en el navegador. En este caso, ya que la respuesta consiste de datos XML, el navegador muestra los datos en su visor XML nativo.



Ajax es simplemente el intercambio de información a través de HTTP, y la actualización dinámica de la página basándose en los resultados.