Proyecto práctico – Desarrollo de un sistema.

Material Teórico

1. UIX / UX Experiencia del usuario
2. Envíos de Mails automatizados
3. Ajax Inicial
4. Ejemplo de Uso Ajax

Material para prácticas de clase

5. FrontEnd y Servlets
6. Relación Servlets y Clases
7. Copia Estructura proyecto.sql
8. Archivo Connector Java - MySql

Proyecto de desarrollo

9. Actividades.html desarrollo de la sección
10. Programación Tarjetas del Blog
11. Bootstrap – Configuración de Columnas
12. Archivo de Código Index.html

Material para prácticas de clase

13. Controlador MySQL en Java. Instalación y Uso
14. Validando usuario y clave
15. Material de prácticas programadas
16. Código Fuente del Proyecto de Usuarios

Proyecto de desarrollo

17. Sección Trabajos Realizados con CSS
18. Sección Contactanos con CSS
19. Código fuente de estilos.css
20. Código fuente de blog.html
21. Aplicando CSS a todo el proyecto

Nota: Así se veían los conceptos de UX / UI hasta que interrumpe en la experiencia de usuario el nuevo concepto de IXD y UIX.

¿Qué es UX?

UX (por sus siglas en inglés User eXperience) o en español Experiencia de Usuario, es aquello que una persona percibeal interactuar con un producto o servicio. Logramos una buena UXal enfocarnos en diseñar productos útiles, usables y deseables, lo cual influye en que el usuario se sienta satisfecho, feliz y encantado.

UX es aquello que una persona percibe al interactuar con un producto o servicio.

Es muy común, que el término UX, se confunda con el de Usabilidad o UI. Pero vamos a dejarlo claro.

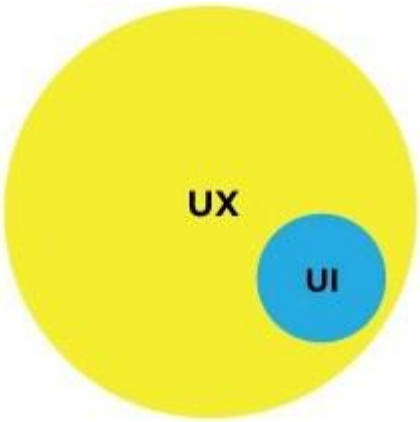
La Usabilidad es un atributo de una buena experiencia de usuario y la UI o Interfaz del Usuario es con lo que se interactúa.

¿Qué es UI?

UI (por sus siglas en inglés User Interface) o en español Interfaz del Usuario, es la vista que permite a un usuario interactuar de manera efectiva con un sistema. Es la suma de una arquitectura de información + elementos visuales + patrones de interacción.

Hace foco en el artefacto, o, dicho de otra manera, en lo que está dentro de la pantalla. Cuando uno diseña interfaces el problema que está resolviendo está en el diseño: selección y distribución de los elementos de la interfaz (ej. textos y campos del formulario), consistencia del diseño (con la plataforma, con otras pantallas), etc. Es importante aclarar que Diseño de Interfaces no equivale a Diseño Gráfico: el diseño de la interfaz puede incluir o no diseño gráfico. Por ejemplo, cuando uno hace un “borrador” o “esquema inicial de un proyecto” está diseñando una interfaz pero no está aplicando diseño gráfico, y cuando uno aplica reglas de estilo a una interfaz está aplicando diseño gráfico pero no está diseñando una interfaz.

Una buena UI nos permite dar una buena UX, pero no lo es todo, es un instrumento.



¿Pero, cómo se logra esta experiencia?

Una buena UX (User Experience) / UIX se logra a través del Diseño Centrado en el Humano, el cual es el enfoque de conocer las necesidades de los usuarios y alinearlos a los objetivos del negocio tomando también en cuenta las limitaciones técnicas.

Hacer foco en el usuario y en la experiencia que se quiere lograr. UX se refiere a lo que *experimenta* el usuario antes, durante y después de interactuar con el artefacto. Sin incorporar al usuario, no se puede hacer UX. Por eso, resulta fundamental en el diseño de la experiencia, comprender en primer lugar a los usuarios y sus verdaderas motivaciones y necesidades, considerar desde ese lugar qué *interfaz,* qué *contenidos* y qué *interacciones* lograrán el resultado buscado, y finalmente, validar con usuarios los resultados que produce la interfaz propuesta.

La persona que realiza UX, es conocida como un UX Designer y como parte de sus responsabilidades está el de investigar qué es lo que las personas necesitan para cumplir sus objetivos y resolver sus inconvenientes.

Dentro de las actividades que realiza un UX Designer, están:

- Investigación (con estadísticas, etnográfica, entrevistas 1 a 1…)
- Evaluación (evaluaciones heurísticas, benchmarks, pruebas de usabilidad)
- Análisis de datos (KPI's, métricas)
- Arquitectura de información

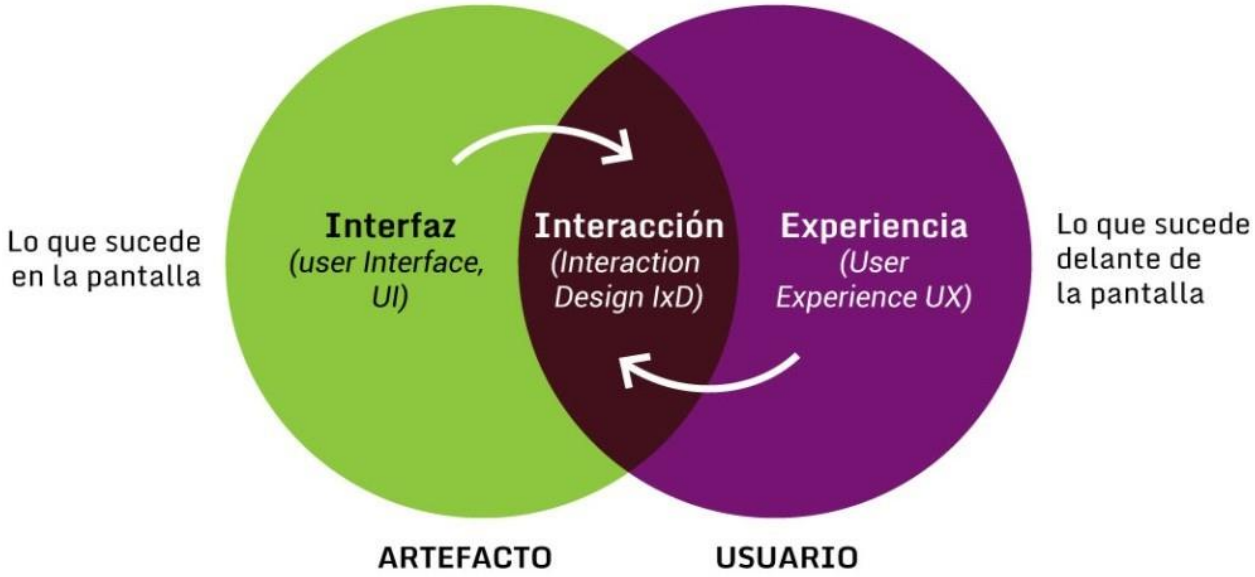
Una persona que se desempeña como UX Designer por lo general tiene conocimientos generales de:

- Psicología, Sociología o Antropología
- Tecnología, Desarrollo de Productos Digitales
- Comunicación, Marketing
- Negocios, Ventas
- Diseño Industrial, Gráfico

El modelo actual, ya no apunta a definir UX y por otro lado UI, sino que ya apunta al reconocimiento de UIX o IXD como camino a seguir. Por lo que la imagen inicial de este documento ya queda fuera de época, pudiendo interpretar que ahora UI sería el mantel y el plato en donde cenamos y el elemento con el que el usuario interactúa con la comida es la cuchara, conformando ésta última el IXD de la experiencia del usuario.

LA NUEVA TENDENCIA.
DISEÑO DE INTERACCIÓN, UIX O IXD

Entonces, la nueva forma de análisis, trata a IXD (Interaction Design) como la forma de definir las formas de operar la interfaz (por ejemplo, si el ingreso de información o selección se produce mediante teclado, mouse, touch, o una combinación de ellos), los flujos de operación y las respuestas del sistema, en el caso de la primer imagen, la cuchara. En definitiva, pone el foco en el contacto entre el usuario y el artefacto.



Por otro lado, UIX, viene a incorporarse al lenguaje habitual como la usabilidad y experiencia de usuario conformada por UI + UX + IXD = UIX.

Envío de mails desde un proyecto Java

1. Introducción
JavaMail se trata de una librería desarrollada por SUN encaminada al envío de correos electrónicos directamente desde tu aplicación Java. El uso de ésta librería es muy sencillo pero detallaremos paso a paso como realizar la instalación y uso de ella.

2. Instalación
En primer lugar debemos realizar la descarga de la librería desde :
https://www.oracle.com/java/technologies/javamail-releases.html

La instalación de la librería para poder hacer uso de ella en nuestro proyecto se puede realizar de 2 maneras diferentes, o bien la importamos desde el entorno de desarrollo que estemos usando (en mi caso la probé en Eclipse y NetBeans) o bien modificamos el CLASSPATH del sistema.

Ejemplo de Importación de la librería con Eclipse:

- Accedemos a las propiedades del proyecto (Boton derecho → propiedades).
- Seleccionamos en el arbol de la izquierda la opción “Java Build Path”y la pestaña “libraries”.
- Entramos en la opción “Add external JARs”, buscamos el archivo “mail.jar” que se incluye dentro del archivo que nos acabamos de bajar.
- Importación de la librería en el CLASSPATH:
- Abrimos una consola de comandos y tecleamos “set CLASSPATH=%CLASSPATH%;C:\javamail\mail.jar“ (sustituyendo la ruta por la vuestra correspondiente).

3. Clases y métodos básicos usados

Clase Properties: Ésta clase es la encargada de almacenar las propiedades de la conexión que vamos a establecer con el servidor de correo Saliente SMTP.

Método Put: Mediante éste método asignaremos las propiedades que necesitamos, como son: Servidor SMTP.Valor booleano de habilitación

- TLS.
- Puerto SMTP
- E-mail emisor del mensaje
- Usuario de acceso al servidor
- SMTP
- Valor booleano de contraseña requerida.

Método Get: Obtención de los parámetros anteriores ya guardados.
Clase Session: Será la clase encargada de manejar la sesión de usuario
Método GetDefaultInstance: e introduciremos la variable de la clase Properties que nos hemos creado anteriormente y ésta nos creará una sesión para dichas propiedades.
Método GetTransport: Indicaremos a éste método el protocolo de transporte a utilizar (en nuestro caso smtp).
Clase MimeMessage: Aquí formaremos el mensaje que deseamos enviar.
Constructor: Se le introduce al constructor la sesión sobre la que vamos a enviar el mensaje

Método SetFrom:Recibe como parámetro la dirección del emisor del mensaje de tipo InternetAddress.
Método AddRecipient: Recibe 2 parámetros, por un lado el tipo de receptor que vamos a especificar descritos en la clase Message.RecipientType (TO,CC,BCC). Como segundo parámetro le pasaremos igual que en el método anterior, una variable de la clase InternetAddress con la dirección del receptor.
Método SetSubject: Introducimos el asunto del mensaje como único parámetro en forma de String.
Método SetText: De igual forma que en el método anterior, introducimos el texto del mensaje en forma de String.
Clase Transport: Define los parámetros del protocolo de transporte. Para empezar, inicializaremos la variable obteniendo el tipo de protocolo de transporte de la clase session explicada anteriormente.
Método Connect: Se encarga de establecer la conexión con el servidor, introduciendo el nombre de usuario y contraseña (si es requerida).
Método SendMessage: Envía el mensaje que hemos creado anteriormente a los destinatarios especificados.
Método Close: Cierra la conexión.

4. Ejemplo básico
Código Java

```
package com.prueba.javamail;

import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class EmailSenderService {
	private final Properties properties = new Properties();

	private String password;

	private Session session;

	private void init() {

		properties.put("mail.smtp.host", "mail.gmail.com");
		properties.put("mail.smtp.starttls.enable", "true");
		properties.put("mail.smtp.port",25);
		properties.put("mail.smtp.mail.sender","emisor@gmail.com");
		properties.put("mail.smtp.user", "usuario");
		properties.put("mail.smtp.auth", "true");

		session = Session.getDefaultInstance(properties);
	}

	public void sendEmail(){
```

```
                init();
                try{
                        MimeMessage message = new MimeMessage(session);
                        message.setFrom(new
InternetAddress((String)properties.get("mail.smtp.mail.sender")));
                        message.addRecipient(Message.RecipientType.TO, new
InternetAddress("receptor@gmail.com"));
                        message.setSubject("Prueba");
                        message.setText("Texto");
                        Transport t = session.getTransport("smtp");
                        t.connect((String)properties.get("mail.smtp.user"), "password");
                        t.sendMessage(message, message.getAllRecipients());
                        t.close();
                }catch (MessagingException me){
                //Aqui se debería o mostrar un mensaje de error o en lugar
                //de no hacer nada con la excepcion, lanzarla para que el modulo
                //superior la capture y avise al usuario algún mensaje.
                        return;
                }

        }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package com.prueba.javamail;

import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class EmailSenderService {
        private final Properties properties = new Properties();

        private String password;

        private Session session;

        private void init() {

                properties.put("mail.smtp.host", "mail.gmail.com");
                properties.put("mail.smtp.starttls.enable", "true");
                properties.put("mail.smtp.port",25);
                properties.put("mail.smtp.mail.sender","emisor@gmail.com");
                properties.put("mail.smtp.user", "usuario");
                properties.put("mail.smtp.auth", "true");

                session = Session.getDefaultInstance(properties);
```

```
	}

	public void sendEmail(){

		init();
		try{
			MimeMessage message = new MimeMessage(session);
			message.setFrom(new
InternetAddress((String)properties.get("mail.smtp.mail.sender")));
			message.addRecipient(Message.RecipientType.TO, new
InternetAddress("receptor@gmail.com"));
			message.setSubject("Prueba");
			message.setText("Texto");
			Transport t = session.getTransport("smtp");
			t.connect((String)properties.get("mail.smtp.user"), "password");
			t.sendMessage(message, message.getAllRecipients());
			t.close();
		}catch (MessagingException me){
		//Aqui se debería o mostrar un mensaje de error o en lugar
		 //de no hacer nada con la excepcion, lanzarla para que el modulo
		 //superior la capture y avise al usuario algún mensaje.
			return;
		}
	}
}
```

¿Qué es AJAX?

AJAX (JavaScript Asíncrono y XML) es un término nuevo para describir dos capacidades de los navegadores que han estado presentes por años, pero que habían sido ignoradas por muchos desarrolladores Web, hasta hace poco que surgieron aplicaciones como Gmail, Google suggest y Google Maps.

Las dos capacidades en cuestión son:

La posibilidad de hacer peticiones al servidor sin tener que volver a cargar la página.
La posibilidad de analizar y trabajar con documentos XML.
Primer Paso – Cómo realizar una petición HTTP al servidor
Para realizar una petición HTTP usando JavaScript, es necesario crear una instancia de una clase que provea esta funcionalidad. Esta clase fue inicialmente introducida en Internet Explorer como un objeto ActiveX, llamado XMLHTTP. Después Mozilla, Safari y otros navegadores lo siguieron, implementando una clase XMLHttpRequest que soportaba los métodos y las propiedades del objeto ActiveX original.

Como resultado, para crear una instancia de la clase requerida que funcione en todos los navegadores, es necesario poner:

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

(El código mostrado es una versión simplificada con fines ilustrativos. Para un ejemplo más completo ver el paso 3.)

Algunas versiones de los navegadores Mozilla no funcionan correctamente si la respuesta del servidor no tiene la cabecera mime de tipo XML. En ese caso es posible usar un método extra que sobreescriba la cabecera enviada por el servidor, en caso que no sea text/xml.

```
http_request = new XMLHttpRequest();
http_request.overrideMimeType('text/xml');
```

El próximo paso es decidir qué se hará después de recibir la respuesta del servidor a la petición enviada. A estas alturas sólo es necesario decirle al objeto HTTPrequest qué función de JavaScript se encargará de procesar la respuesta. Para esto se asigna la propiedad onreadystatechange del objeto al nombre de la función de JavaScript que se va a utilizar:

```
http_request.onreadystatechange = nameOfTheFunction;
```

Es importante notar que no hay paréntesis después del nombre de la función y no se pasa ningún parámetro. También es posible definir la función en ese momento, y poner en seguida las acciones que procesarán la respuesta:

```
http_request.onreadystatechange = function(){
    // procesar la respuesta
};
```

Después de especificar que pasará al recibir la respuesta es necesario hacer la petición. Para esto se utilizan los métodos open() y send() de la clase HTTP request, como se muestra a continuación:

```
http_request.open('GET', 'http://www.example.org/algun.archivo', true);
http_request.send();
```

El primer parámetro de la llamada a open() es el método HTTP request – GET, POST, HEAD o cualquier otro método que se quiera usar y sea aceptado por el servidor. El nombre del método se escribe en mayúsculas, sino algunos navegadores (como Firefox) podrían no procesar la petición.

El segundo parámetro es el URL de la página que se esta pidiendo. Por medida de seguridad no es posible llamar páginas en dominios de terceras personas. Se debe saber el dominio exacto de todas las páginas o se obtendrá un error de 'permiso denegado' al llamar open(). Una falla común es acceder al sitio por domain.tld e intentar llamar las páginas como www.domain.tld.
El tercer parámetro establece si la petición es asíncrona. Si se define TRUE, la ejecución de la función de JavaScript continuará aún cuando la respuesta del servidor no haya llegado. Por esta capacidad es la A en AJAX.
El parámetro en el método send()puede ser cualquier información que se quiera enviar al servidor si se usa POST para la petición. La información se debe enviar en forma de cadena, por ejemplo:

```
name=value&anothername=othervalue&so=on
```

Si se quiere enviar información de esta forma, es necesario cambiar el tipo MIME de la petición usando la siguiente línea:

```
http_request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```
De otro modo el servidor descartará la información.

Segundo Paso – Procesando la respuesta del servidor
Al enviar la petición HTTP es necesario indicar el nombre de la función JavaScript que procesará la respuesta.

```
http_request.onreadystatechange = nameOfTheFunction;
```

A continuación se verá lo que esta función realiza. En primer lugar necesita revisar el estado de la petición. Si el estado tiene el valor 4, significa que la respuesta completa del servidor ha sido recibida y es posible continuar procesándola.

```
if (http_request.readyState == 4) {
    // todo va bien, respuesta recibida
} else {
    // aun no esta listo
}
```
La lista completa de valores para la propiedad readyState es:

0 (no inicializada)
1 (leyendo)
2 (leido)
3 (interactiva)
4 (completo)
(Source)

Ahora es necesario revisar el código de status de la respuesta HTTP. La lista completa de códigos aparece en el sitio de la W3C. Para el próposito de este artículo sólo es importante el código 200 OK.

```
if (http_request.status == 200) {
   // perfectp!
} else {
   // hubo algún problema con la petición,
   // por ejemplo código de respuesta 404 (Archivo no encontrado)
   // o 500 (Internal Server Error)
}
```

Después de haber revisado el estado de la petición y el código de status de la respuesta, depende de uno hacer cualquier cosa con la información que el servidor ha entregado. Existen dos opciones para tener acceso a esa información:

http_request.responseText – regresará la respuesta del servidor como una cadena de texto.
http_request.responseXML – regresará la respuesta del servidor como un objeto XMLDocument que se puede recorrer usando las funciones de JavaScript DOM.

Tercer Paso – Un sencillo ejemplo

En este ejemplo se utilizará todo lo que se ha visto para hacer una petición HTTP. Se pedirá un documento HTML llamado test.html, que contiene el texto "Esto es una prueba." y después usaremos la función alert() con el contenido del archivo test.html .

```
<script type="text/javascript" language="javascript">

   var http_request = false;

   function makeRequest(url) {

      http_request = false;

      if (window.XMLHttpRequest) { // Mozilla, Safari,...
         http_request = new XMLHttpRequest();
         if (http_request.overrideMimeType) {
            http_request.overrideMimeType('text/xml');
            // Ver nota sobre esta linea al final
         }
      } else if (window.ActiveXObject) { // IE
         try {
            http_request = new ActiveXObject("Msxml2.XMLHTTP");
         } catch (e) {
            try {
               http_request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {}
         }
      }

      if (!http_request) {
         alert('Falla :( No es posible crear una instancia XMLHTTP');
         return false;
      }
```

```
        http_request.onreadystatechange = alertContents;
        http_request.open('GET', url, true);
        http_request.send();

    }

    function alertContents() {

        if (http_request.readyState == 4) {
            if (http_request.status == 200) {
                alert(http_request.responseText);
            } else {
                alert('Hubo problemas con la petición.');
            }
        }

    }
</script>
<span
    style="cursor: pointer; text-decoration: underline"
    onclick="makeRequest('test.html')">
        Hacer una petición
</span>
```

En este ejemplo:

El usuario presiona el vínculo "Hacer una petición" en el navegador;
Esto llama la función makeRequest() que tiene como parámetro test.html que es un archivo HTML localizado en el mismo directorio;
La petición es realizada y después (onreadystatechange) la ejecución pasa a alertContents();
alertContents() verifica si la respuesta fue recibida y si es OK, si es así utiliza alert() con el contenido de test.html.

Cuarto Paso – Trabajando con la respuesta XML
En el ejemplo anterior se utilizo la propiedad reponseText del objeto pedido para mostrar el contenido de test.html una vez que la respuesta HTTP había sido recibida. En éste se utilizará la propiedad responseXML.

Primero hay que crear un documento de XML válido. El documento (test.xml) contiene lo siguiente:

```
<?xml version="1.0" ?>
<root>
    Esto es una prueba.
</root>
```

Para que funcione el script solo es necesario cambiar la línea de petición por:

```
...
onclick="makeRequest('test.xml')">
...
```

Y en alertContents() es necerario remplazar la línea donde aparece alert(http_request.responseText); por:

```
var xmldoc = http_request.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```