DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 05

Implementación con Socket.io

Alumno(s):		***Nota***	
Grupo:		***Ciclo:V***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Entiende cómo funciona socket.io					
Utiliza mongoose para conexión de base de datos					
Desarrolla aplicaciones web con express					
Realiza con éxito lo propuesto en la tarea.					
Es puntual y redacta el informe adecuadamente					

Laboratorio 05:
Implementación con Socket.io

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de Socket.io
- Desarrollar aplicaciones web con el framework Express
- Implementación correcta de mongoose

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

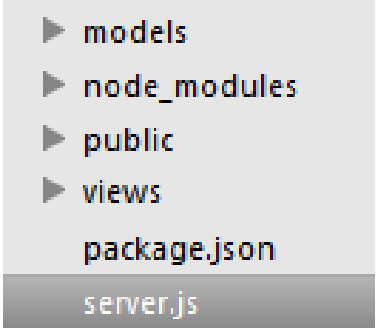
Procedimiento:

Lab Setup

1. **Continuación de laboratorio anterior**
   1.1. En el laboratorio anterior usted configuró una máquina virtual con acceso a Internet, nodeJS y MongoDB. Usará la misma instancia para nuestra sesión actual.
   1.2. Este proyecto necesitará hacer uso de las librerías express, jade, socket.io y mongoose. Cree una carpeta con el nombre lab05 donde copiará el manifiesto proveído por el docente (package.json) y luego, mediante la consola de comandos y ubicándonos en dicha carpeta, ejecutaremos lo siguiente para dejar el proyecto listo para iniciar.

```
>npm install
```

   1.3. Asegurémonos de tener la estructura correcta de proyecto. Cree las carpetas listadas a continuación para tener la disposición adecuada.



   1.4. En la raíz de nuestro proyecto, crearemos el archivo server.js con el siguiente contenido

```
var express = require('express'),
app = express(),
http = require('http').Server(app),
port = process.env.PORT || 3000;

app.set('view engine','jade');

app.use('/static', express.static('public'));

app.get('/', function(req, res){
    res.render('main');
});

http.listen(port, function(){
    console.log('Servidor conectado en *:' + port);
});
```

1.5. Dentro de la carpeta views, crearemos el archivo home.jade con el siguiente contenido

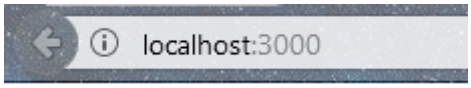
```
doctype
html( lang="es" )
    head
        title APP de Ejemplo
        meta( charset='utf-8' )
        meta( http-equiv='X-UA-Compatible', content='IE=edge' )
        meta( name='viewport', content='width=device-width, initial-scale=1.0' )
        meta( name='description', content='Baking Bootstrap Snippets with Jade' )
        link(href="//maxcdn.bootstrapcdn.com/bootswatch/3.3.0/flatly/bootstrap.min.css", rel="stylesheet")

    body(style="padding-bottom:10rem;")
        .container
            block content
        script( src='//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js' )
        script( src='//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/bootstrap.min.js' )
```

1.6. Dentro de la carpeta views, crearemos el archivo main.jade con el siguiente contenido

```
extends home
block content
    h1 Ejemplo de CRUD con Socket IO
    .col-xs-4
        form#formulario
            .form-group
                label(for="_id") ID
                input#_id.form-control(type="text",placeholder="ID del usuario")
            .form-group
                label(for="first_name") Nombres
                input#first_name.form-control(type="text",placeholder="Su nombre")
            .form-group
                label(for="last_name") Apellidos
                input#last_name.form-control(type="text",placeholder="Sus apellidos")
            .form-group
                label(for="timezone") Regi&oacute;n
                input#timezone.form-control(type="text",placeholder="-5",value="-5")
            .form-group
                label(for="locale") Idioma
                input#locale.form-control(type="text",placeholder="Idioma de preferencia")
            .form-group
                label(for="profile_pic") Foto
                input#profile_pic.form-control(type="text",placeholder="URL de foto")
            button.btn.btn-default(type="submit") Guardar
    .col-xs-8
        table.table.table-striped.table-bordered.table-hover
            thead
                tr
                    th ID
                    th Nombres
                    th Apellidos
                    th Zona Horaria
                    th Idioma
                    th Foto
            tbody
```

1.7. Iniciemos nuestro servicio y veamos el resultado del mismo. Adjunte una captura de lo obtenido.



2. Creación de registro con mongoose y socket.io

2.1. Capturaremos el envío del formulario para redirigirlo a nuestro socket con socket.io Para eso utilizaremos el evento submit del formulario ya declarado.

2.2. Crearemos el archivo app.js dentro de la carpeta js que será creada dentro de la carpeta public, con el siguiente contenido

```
$(document).ready(function(){
    var socket = io();
    $('#formulario').submit(function(e){
        e.preventDefault();
        var data = {
            _id: $('#_id').val(),
            first_name: $('#first_name').val(),
            last_name: $('#last_name').val(),
            timezone: $('#timezone').val(),
            locale: $('#locale').val(),
            profile_pic: $('#profile_pic').val()
        };
        if(data._id==''){
            $('#_id').focus();
            return alert('Debe ingresar un ID!');
        }
        if(data.first_name==''){
            $('#first_name').focus();
            return alert('Debe ingresar un nombre!');
        }
        socket.emit('crear',data);
        $('#formulario').trigger('reset');
        return true;
    });
});
```

2.3. Agreguemos las siguientes líneas a nuestro archivo home.jade

```
script( src='//cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.2/socket.io.js' )
script( src='static/js/app.js' )
```

2.4. Agregaremos las siguientes líneas al archivo server.js

```
io = require('socket.io')(http)
```

```
io.on('connection', function(socket){
    console.log('Usuario conectado!');
    socket.on('disconnect', function () {
        console.log('Usuario desconectado!');
    });
});
```

2.5. Reinicie su servidor node y actualice su navegador. La variable io declarada hacer un momento corresponde al manejador de sockets acoplado a nuestra aplicación. Sobre esta variable declararemos el evento “connection” que como indica su nombre, se disparará al momento de que un cliente se conecte al servicio. Esto lo verificamos con la línea `console.log('Usuario conectado!');` que nos avisará de dicho evento. Así mismo, si cerramos el navegador, el socket se cerrará automáticamente y se disparará el evento “disconnect” del socket iniciado. Agregue una captura de lo mostrado en su consola después de estas pruebas.

2.6. Hablemos un poco del código introducido. En app.js, `socket.emit('crear',data);` emite un evento a nuestro socket, el cual se llama crear y se le envía un objeto con los datos recogidos es nuestro formulario. Ahora mismo no hemos declarado dicho evento de escucha en nuestro servidor por lo que es necesario declararlo. Agregue esta porción de código antes de nuestro evento de desconexión en server.js

```
socket.on('crear',function(data){
    console.log(data);
});
```

2.7. Reinicie el servidor y haga la prueba de enviar los datos del formulario. Deberíamos ver una respuesta en la consola. Adjuntar captura.

2.8. Vemos que estamos recibiendo exitosamente los datos del formulario, es hora de conectarlo con MongoDB. Creemos el archivo user.js dentro de la carpeta models con el siguiente contenido.

```
var mongoose = require('mongoose'),
Schema = mongoose.Schema;

mongoose.connect('mongodb://localhost/chat');

var user_schema = new Schema({
    _id: String,
    first_name: String,
    last_name: String,
    timezone: String,
    locale: String,
    profile_pic: String
});
user_model = mongoose.model('user', user_schema,'users');

module.exports = {
    create: function(data,callback){
        var item = {
            _id: data._id,
            first_name: data.first_name,
            last_name: data.last_name,
            timezone: data.timezone,
            locale: data.locale,
            profile_pic: data.profile_pic
        };
        var nuevo = new user_model(item).save();
        callback(item);
    }
};
```

2.9. Modifiquemos el listener llamado crear de nuestro socket en el archivo server.js para que tenga la siguiente forma

```
socket.on('crear',function(data){
    user.create(data,function(rpta){
        io.emit('nuevo',rpta);
    });
});
```

2.10. No se olvide de incluir el archivo user.js dentro del server.js para su correcta invocación. Agregue la siguiente línea.

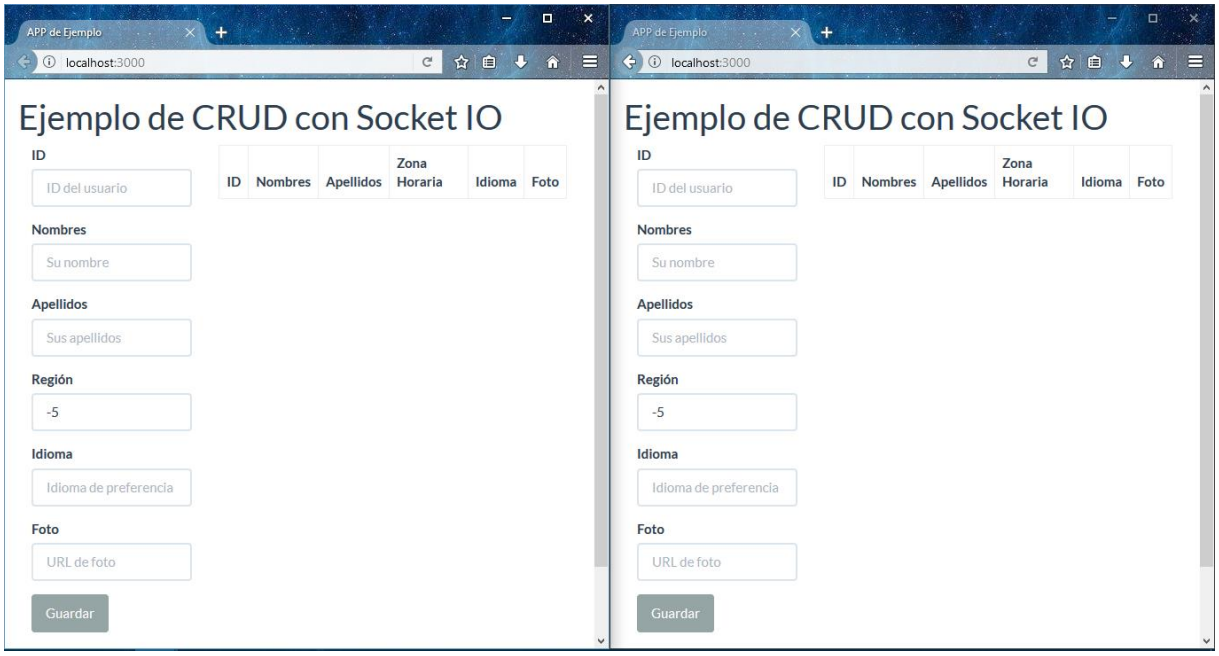
```
user = require('./models/user');
```

2.11. Nuestro evento crear llamará al modelo user e insertará los datos recibidos del formulario. Al finalizar este evento, se disparará un callback (función de retorno) que usará el método emit del objeto io. Emit nos permite emitir información a nuestros clientes conectados.

Ellos escucharán de acuerdo al nombre de evento que declaremos, en este caso, hemos declarado el evento nuevo, por lo que debemos declararlo en nuestro archivo app.js. Agreguemos las siguientes líneas después de la declaración de nuestra variable socket. Observa que así mismo, estamos declarando una función fill, cuyo objetivo será crear una fila con la información obtenida para mostrar al usuario.

```
socket.on('nuevo',function(data){
    fill(data);
});
var fill = function(data){
    var $row = $('<tr id="'+data._id+'">');
    $row.append('<td>'+data._id+'</td>');
    $row.append('<td>'+data.first_name+'</td>');
    $row.append('<td>'+data.last_name+'</td>');
    $row.append('<td>'+data.timezone+'</td>');
    $row.append('<td>'+data.locale+'</td>');
    $row.append('<td>'+data.profile_pic+'</td>');
    $row.append('<td><button class="btn btn-success btn-sm" name="btnAct">Actualizar</button></td>');
    $row.append('<td><button class="btn btn-danger btn-sm" name="btnEli">Eliminar</button></td>');
    $row.data('data',data);
    $('table tbody').append($row);
};
```

2.12. Se recomienda tener dos pantallas para observar de forma notoria los cambios realizados. Proceda a abrir dos ventanas de navegador y apílelas según su conveniencia, se adjunta una recomendación para que usted pueda comprobar que los cambios realizados en una ventana se reflejan en la otra.



2.13. Reinicie el servidor node y refresque el navegador. Envíe su formulario y vea lo que sucede con su grilla principal. Así mismo, verifique que este cambio se refleja en la otra ventana.

2.14. Así mismo, adjunte una captura de que dicho registro se haya almacenado en su base de datos.

3. Listado Inicial de información

3.1. El problema que ahora se nos presenta es que si refrescamos uno de los clientes, dejará de mostrar en su grilla la data ingresada, a pesar que esta sigue existiendo en la base de datos. Procederemos a crear un método listar que se ejecutará cada vez que se conecte un usuario. Agreguemos lo siguiente en el server.js, antes de nuestro listener del evento crear.

```
user.show(function(data){
    io.emit('listar',data);
});
```

3.2. Como hemos emitido el evento listar, debemos poner un escucha en el cliente para su respectivo tratamiento. Agregamos las siguientes líneas al app.js

```
socket.on('listar',function(data){
    data = JSON.parse(data);
    for(var i=0,j=data.length; i<j; i++){
        fill(data[i]);
    }
});
```

3.3. Finalmente, añadimos la función show a nuestro modelo user.

```
show: function(callback){
    user_model.find({},function(err,items){
        if (!err) {
            callback(JSON.stringify(items));
        }else{
            return console.log(err);
        }
    });
},
```

3.4. Reinicie el servidor y luego refresque un navegador de los dos abiertos. Vea el resultado y adjunte una captura.

3.5. Ahora refresque el otro navegador y vea lo que sucede en el primero de los dos navegadores. Describa lo sucedido.

3.6. Esto sucede debido a que el emit del objeto io se encarga de transmitir una noticia a todos nuestros clientes conectados. Es nuestra labor discernir en que momentos usaremos una transmisión a todos los usuarios y cuando una transmisión a la persona que tenemos conectada en el socket. Reemplace lo agregado recientemente en server.js por lo siguiente

```
user.show(function(data){
    socket.emit('listar',data);
});
```

3.7. Reinicie el servidor y luego refresque un navegador de los dos abiertos. Vea el resultado y adjunte una captura.

4. Actualizar información en base de datos y en interfaz

4.1. Agregaremos el evento click al botón actualizar que se genera con cada registro. Agregamos el siguiente código dentro de nuestra función fill, antes de la inserción de nuestro objeto $row al tbody de la tabla.

```
$row.find('[name=btnAct]').click(function(){
    var data = $(this).closest('tr').data('data');
    $('#_id').val(data._id);
    $('#first_name').val(data.first_name);
    $('#last_name').val(data.last_name);
    $('#timezone').val(data.timezone);
    $('#locale').val(data.locale);
    $('#profile_pic').val(data.profile_pic);
    $('.warning').removeClass('warning');
    $(this).closest('tr').addClass('warning');
});
```

4.2. La línea `$('.warning').removeClass('warning');` será utilizada con dos propósitos: el más obvio es para cambiar el color de la fila y mostrarle al usuario qué cosa está editando, mientras que la segunda razón será que nuestro formulario al momento de enviar la data, se

dé cuenta que se trata de una edición de un registro y no de la creación de uno nuevo. Reemplazaremos la línea `socket.emit('crear',data);` ubicada dentro del submit de nuestro formulario en app.js por lo siguiente

```
var accion = 'crear';
if($('.warning').length>0) accion = 'actualizar';
$('.warning').removeClass('warning');
socket.emit(accion,data);
```

4.3. Como recibiremos nuevamente un registro desde el servidor, debemos verificar si este ser agregado (en el momento de creación) o actualizado (en el caso de modificación) de nuestra grilla. Modifique la función fill de tal forma que luzca de la siguiente manera

```
var fill = function(data){
    if($('#'+data._id).length==0){
        var $row = $('<tr id="'+data._id+'">');
        $row.append('<td>'+data._id+'</td>');
        $row.append('<td>'+data.first_name+'</td>');
        $row.append('<td>'+data.last_name+'</td>');
        $row.append('<td>'+data.timezone+'</td>');
        $row.append('<td>'+data.locale+'</td>');
        $row.append('<td>'+data.profile_pic+'</td>');
        $row.append('<td><button class="btn btn-success btn-sm" name="btnAct">Actualizar</button></td>');
        $row.append('<td><button class="btn btn-danger btn-sm" name="btnEli">Eliminar</button></td>');
        $row.data('data',data);
        $row.find('[name=btnAct]').click(function(){
            var data = $(this).closest('tr').data('data');
            $('#_id').val(data._id);
            $('#first_name').val(data.first_name);
            $('#last_name').val(data.last_name);
            $('#timezone').val(data.timezone);
            $('#locale').val(data.locale);
            $('#profile_pic').val(data.profile_pic);
            $('.warning').removeClass('warning');
            $(this).closest('tr').addClass('warning');
        });
        $('table tbody').append($row);
    }else{
        var $row = $('#'+data._id);
        $row.find('td:eq(1)').html(data.first_name);
        $row.find('td:eq(2)').html(data.last_name);
        $row.find('td:eq(3)').html(data.timezone);
        $row.find('td:eq(4)').html(data.locale);
        $row.find('td:eq(5)').html(data.profile_pic);
    }
};
```

4.4. Agregaremos el evento de escucha actualizar dentro de nuestro server.js, a la misma altura que nuestro evento crear. Nótese como en este caso nos interesa usar io y no socket al momento de emitir una respuesta. Esto es debido a que como estamos actualizando un registro de la base de datos, nos interesa que esto se comunique a todos los clientes conectados.

```
socket.on('actualizar',function(data){
    user.update(data,function(rpta){
        io.emit('nuevo',rpta);
    });
});
```

4.5. Necesitamos agregar la función update dentro de nuestro modelo user.js con su respectivo callback

```
update: function(data,callback){
    user_model.findOne({_id: data._id},function(err,item){
        item.first_name = data.first_name;
        item.last_name = data.last_name;
        item.timezone = data.timezone;
        item.locale = data.locale;
        item.profile_pic = data.profile_pic;
        item.save();
        callback(item);
    });
},
```

4.6. Reinicie su servidor node y refresque ambos navegadores. Verifique que al actualizar un registro esto se refleja en el otro navegador.

5. Eliminación de registros.

5.1. Para la etapa final del laboratorio, debemos lograr la eliminación de un registro. Añadiremos el evento click a nuestro botón eliminar. Agregamos el siguiente código dentro de nuestra función fill, antes de la inserción de nuestro objeto $row al tbody de la tabla.

```
$row.find('[name=btnEli]').click(function(){
    var _id = $(this).closest('tr').attr('id');
    if (confirm('Continuar con eliminacion de registro?')) {
        socket.emit('eliminar',_id);
    }
});
```

5.2. Agregue la función delete dentro de nuestro modelo user.js

```
delete: function(_id,callback){
    user_model.findOne({_id: _id},function(err,post){
        post.remove();
        callback(_id);
    });
}
```

5.3. En nuestro cliente, debemos añadir un escuchar a la función borrado (la que nos indicará que fila eliminar en nuestra interfaz). Añada el evento al app.js

```
socket.on('borrado',function(data){
    $('#'+data).remove();
});
```

5.4. Finalmente, creamos el escucha y nuestro evento emitir en el server.js, el cuál llamará al modelo user y realizará el borrado del registro.

```
socket.on('eliminar',function(data){
    user.delete(data,function(rpta){
        io.emit('borrado',rpta);
    });
});
```

5.5. Reinicie su servidor node y refresque ambos navegadores. Verifique que al eliminar un registro esto se refleja en el otro navegador.

6. **Finalizar la sesión**

6.1. Apagar el equipo virtual

6.2. Apagar el equipo

Tarea

Mejore el código del laboratorio añadiendo las siguientes funcionalidades:

- Debe aparecer una alerta (notificación) al usuario cuando un registro ha sido modificado (así sabrá que se actualizó desde otro cliente y se está reflejando eso en su navegador)
- Mediante jQuery, añadir un efecto para que el texto de una fila modificada recientemente (tanto creada como actualizada) parpadee de un color a otro, mostrándonos que registro ha variado.
- Modificar el formulario para que acepte un combobox (select) con las opciones Si y No, y que estas se guarden en la base de datos como un booleando equivalente a True y False (respectivamente)

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.