

# MVC

## 1.- Definición

**Modelo-Vista-Controlador (MVC)** es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

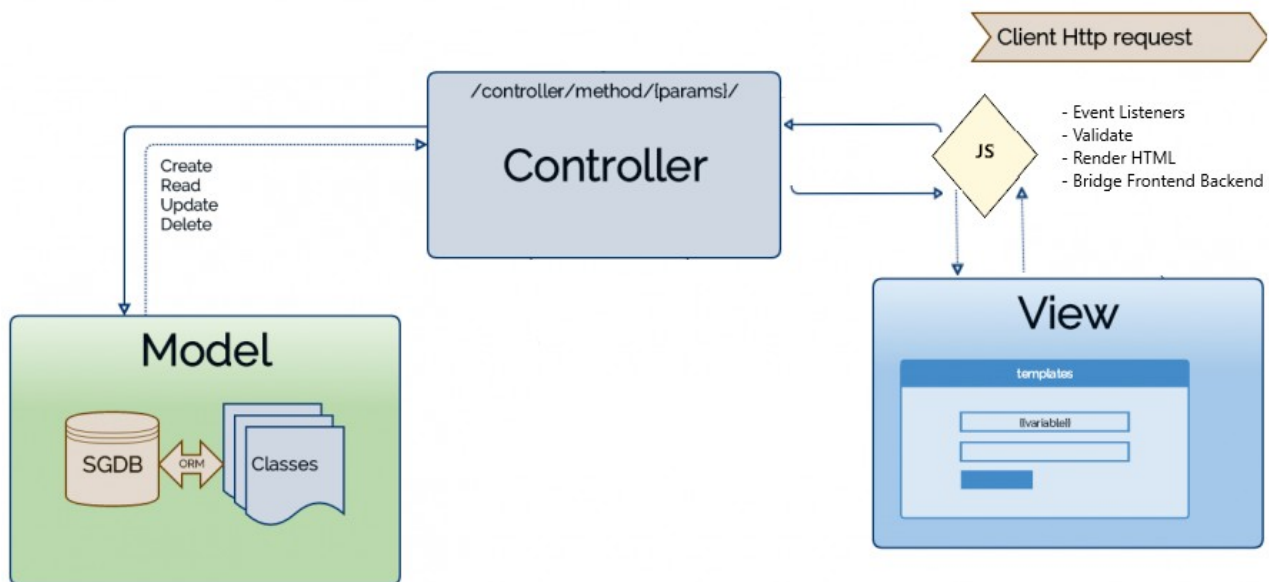
## 2.- Descripción del patrón.

**De manera genérica, los componentes de MVC se podrían definir como sigue:**

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación lógica de negocio. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

## 3.- Objetivo

El objetivo principal de este patrón o cualquier otro similar es de hacer los proyectos más escalables, permitiendo la reutilización de código y una mejor estructuración. Todo ello para facilitar la futura inclusión de nuevas funcionalidades así como para modificar las ya existentes reduciendo la *deuda tecnológica* que podamos adquirir con malas prácticas.



## Interacción de los componentes

Aunque se pueden encontrar diferentes implementaciones de **MVC**, el flujo de control que se sigue generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz- vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (**handler**). ( En nuestro caso serían los ficheros **JavaScript** que actúan de enlace entre el Frontend y el Backend )
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario).
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). *Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.*
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente...

# MVC – IMPLEMENTACIÓN

- Aunque en principio es un patrón de diseño simple puede haber múltiples formas de implementarlo, dependiendo del tipo de aplicación a crear, entorno, objetivos etc.
- Vamos a seguir una posible implementación sin meternos a fondo en la parte técnica de lo que sería una mini-aplicación muy simple con una vista que permite hacer las operaciones básicas de BD (CRUD) sobre una tabla de usuarios.
  - Crear nuevos usuarios.
  - Listar usuarios.
  - Editar usuarios.
  - Borrar usuarios.
- El entorno en este caso es una aplicación web usando los lenguajes:
  - **Frontend** (HTML , CSS , JS) apoyándonos en el framework Bootstrap para el diseño.
  - **Backend** (PHP , MySQL)

Vamos a tener 2 partes diferenciadas en la vista , la parte de la izquierda para registrar nuevos usuarios, la parte de la derecha para el resto de operaciones ,búsqueda, edición, borrado de usuarios.

## Example - MVC-BD

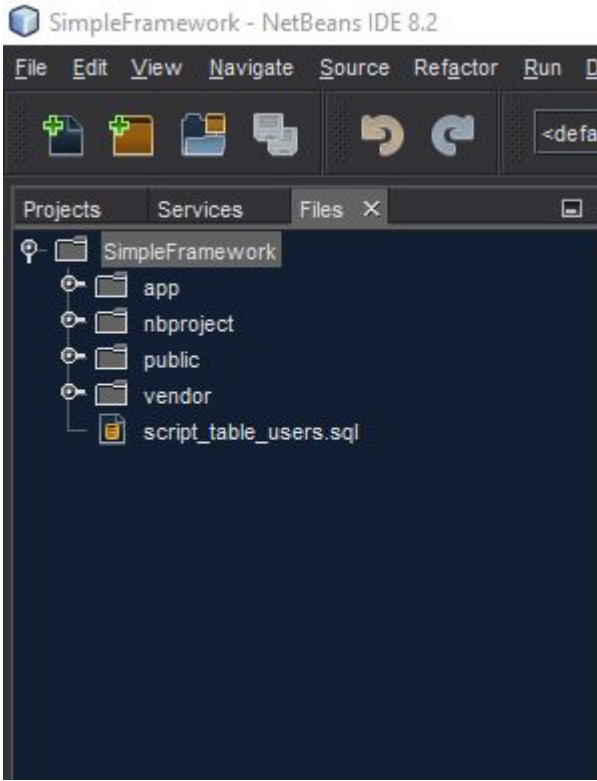
### Formulario Registro

Name	Email	
<input type="text"/>	<input type="text"/>	
Nick	Password	
<input type="text"/>	<input type="password"/>	
Address		
<input type="text"/>		
City	State	Zip Code
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="button" value="Registrar Usuario"/>		

### Listado de Usuarios

<input type="text"/>					<input type="button" value="Buscar"/>
Id	Name	Nick	Email	State	
1	Emerson Beard	eme	emersonbeard@ecolight.com	New York	
2	Tabatha Matthews	tab	tabathamattthews@omnigog.com	Wisconsin	

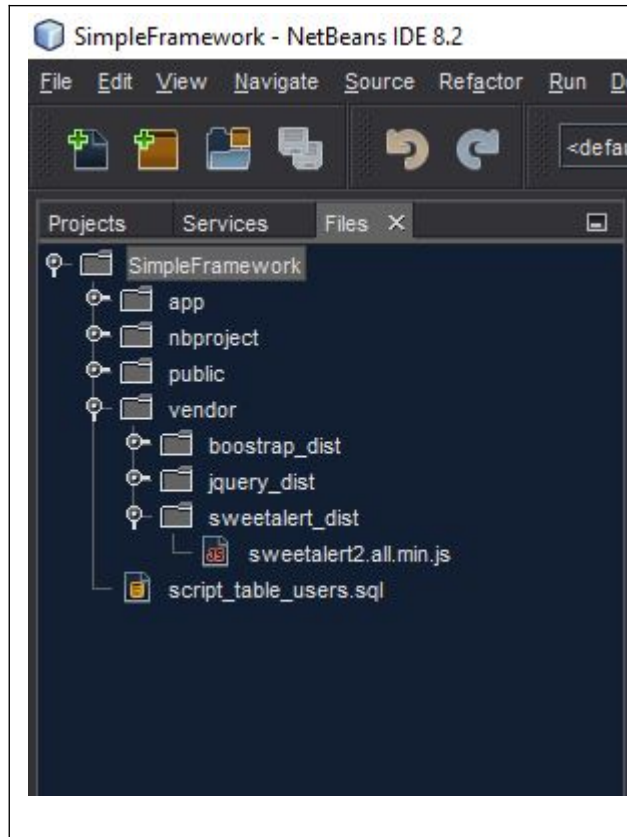
## - Estructura principal de los directorios del proyecto.

	<p>Todos los directorios se subdividen a la vez en otros, como iremos viendo.</p> <ul style="list-style-type: none"><li>• <b>_app</b> =&gt; directorio para los ficheros de configuración, controladores, modelos. Es el núcleo de la aplicación, en donde tendremos como vamos a ver más adelante nuestra estructura MVC.</li><li>• <b>_public</b> =&gt; directorio para acceder a la app en sí (index.php o punto de entrada) y para los assets (ficheros propios de css, js, imágenes etc.)</li><li>• <b>_vendor</b> =&gt; para guardar librerías de terceros que usemos, por ejemplo bootstrap, jquery, sweetalert, carbon etc.</li></ul>
--	---

- En este caso obviaos la carpeta **nbproject** que es una carpeta de configuración interna que crea NetBeans para el proyecto en sí. Y el fichero de script\_table\_users.sql que es simplemente un script creado para generar la tabla que se usa en esta app en concreto.

- Como ya hemos dicho, esta es una posible implementación que vamos a ir desgranando pero por supuesto no es la única, esto siempre va a depender de lo que queramos implementar.

## Estructura de la carpeta vendor.

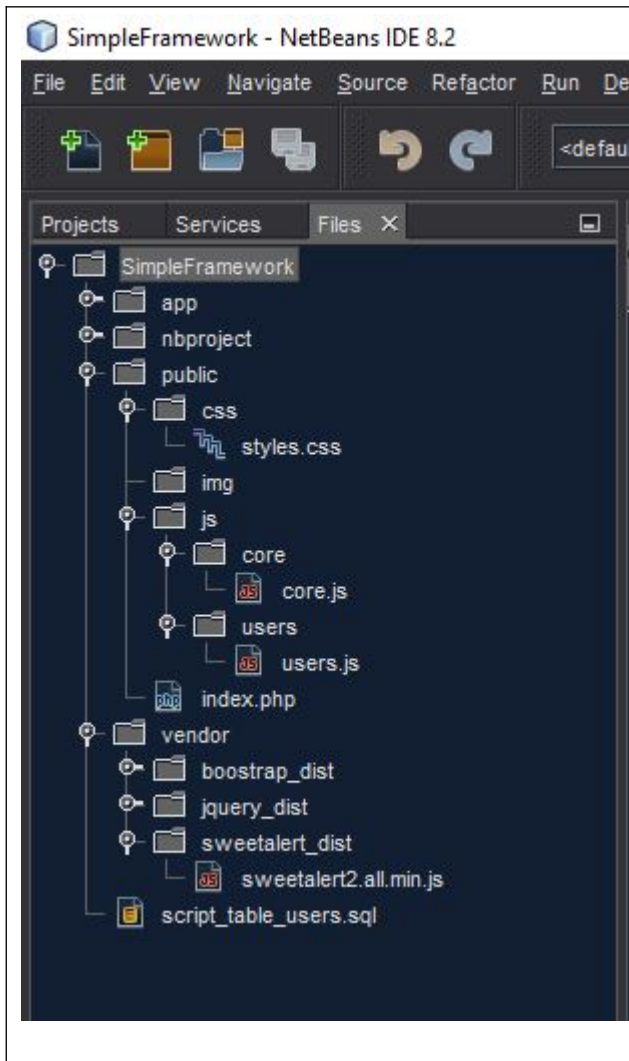


No hay mucho que comentar en esta parte, simplemente aquí guardaremos los ficheros js, css etc. que usen las librerías que tengamos de terceros.

Hay que tener en cuenta que siempre que se pueda estén minificados , y también hay que recordar que a veces puede ser útil acceder a estas librerías si es posible a través de un enlace externo , cdn, cloud...

- Aunque se escapa del objetivo de explicar el patrón MVC , se puede destacar también que esta es la estructura que sigue por ejemplo el gestor de dependencias *composer* para incluir librerías de terceros en nuestro proyecto.

## Estructura de la carpeta public.



En este caso vemos 3 carpetas principales

- **\_css** => aquí guardamos las hojas css propias que usemos en el proyecto.
- **\_img** => para guardar imágenes.
- **\_js** => para los ficheros .js propios que usemos, en nuestro caso vemos 2 subdirectorios.

**\_core:** para funciones .js que sean genéricas y que usemos en varios sitios del proyecto como por ejemplo validaciones, peticiones ajax etc.

**\_users:** en nuestro caso este es el único ambito funcional que vamos a tener, en este .js .Funciones de uso para gestionar los eventos que se produzcan desde la vista, renderizar html, validaciones específicas de este ámbito etc.

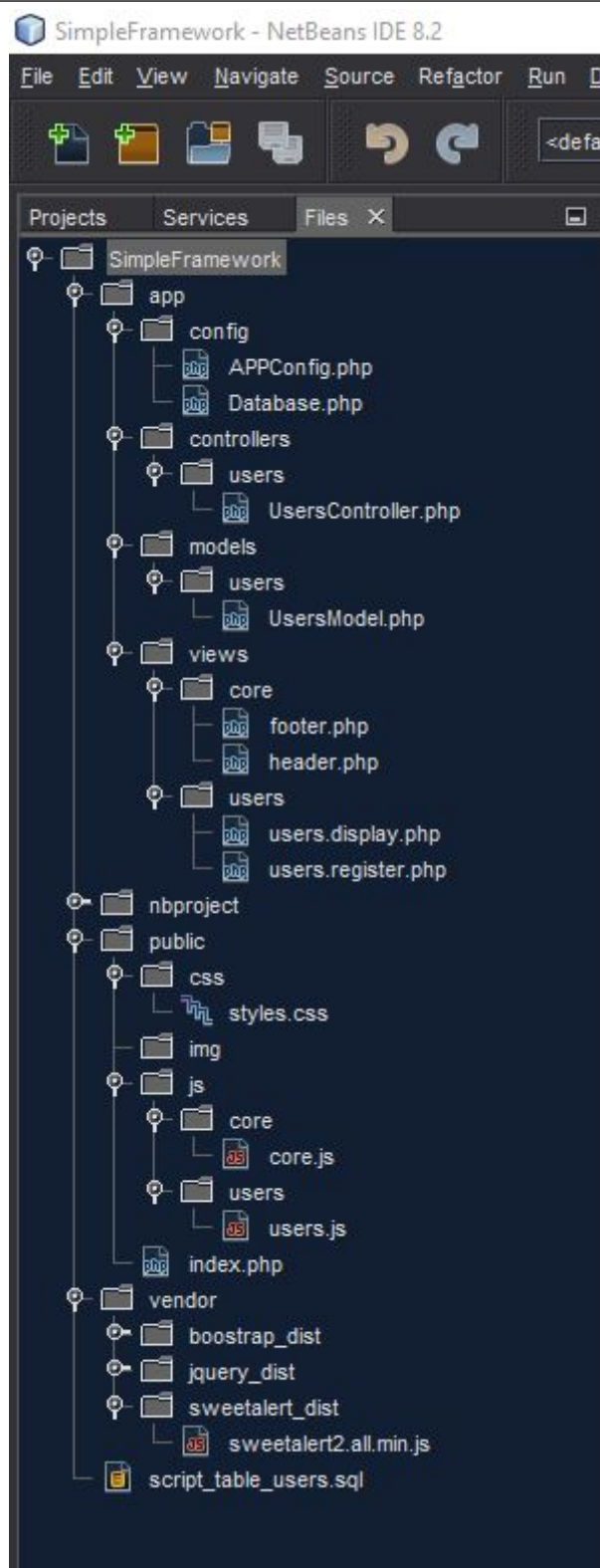
Ya hemos dicho que hay muchas formas de implementar este patrón de diseño, todo depende del proyecto en sí. Otras variaciones por ejemplo es incluir los directorios css, img dentro de un directorio assets.

En public también incluiríamos directorios de ficheros como documentos que usen o genere la aplicación , pdf, xls etc.

Los ámbitos funcionales que tenga la aplicación son muy dependientes de esta claro está. No es lo mismo un ejemplo básico como este que hacer un ERP “*bien estructurado*” que cuente con módulos de contabilidad, RR.HH. , analítica de datos, gestión compras-ventas, facturación etc.

En cada caso tendremos que adaptar nuestra estructura a lo que queramos hacer y teniendo en cuenta futuras extensiones para añadir funcionalidades distintas.

## Estructura de la carpeta app.



Dentro de app en nuestro caso tenemos:

- **\_config** => este directorio lo usaremos para ficheros de configuración como por ejemplo datos de conexión a la BD, constantes que usemos en la aplicación..
- **\_controllers** => aquí dividimos por ambito funcional el código que se encarga de recibir datos desde la vista (en nuestro caso vía JavaScript) y conectar con el modelo que corresponda. Hay veces que este modelo retornará datos que renderizaremos en la vista (caso de un SELECT por ejemplo) y otras ocasiones solo retornará si un INSERT , UPDATE.. se ha efectuado de forma correcta.
- **\_models** => nuestras clases que se encargarán de acceder a la BD, devolverán una respuesta al controlador correspondiente para que este los devuelva a su vez a la vista.
- **\_views** => las vistas en sí, el código html que mostraremos al usuario. En este caso dividimos en 2 carpetas:

**\_core**: también la podríamos llamar de otra forma como layouts. Aquí están las plantillas genéricas que se usan en la app com por ejemplo footer, header, navbar..

**\_users**: codigo html dividido por ámbito funcional, en nuestro caso tenemos 2, *users.display.php* que se correspondería al listado de usuarios de la parte derecha y *users.register.php* que se correspondería con la parte izquierda del formulario de registro.

## EJEMPLO - FLUJO DE EJECUCIÓN

Siguiendo con nuestra mini-aplicación de ejemplo , vamos a seguir un flujo de ejecución como por ejemplo a la hora de registrar un nuevo usuario

1º.- La **vista** tal y como vemos abajo genera un evento cuando hemos rellenado el formulario y pulsamos sobre el botón de Registrar Usuario

### Formulario Registro

Name	Email	
<input type="text" value="Alberto"/>	<input type="text" value="alberto@email.com"/>	
Nick	Password	
<input type="text" value="ATM"/>	<input type="password" value="123qwe!"/>	
Address		
<input type="text" value="Plaza de Manuel Azaña, nº3"/>		
City	State	Zip Code
<input type="text" value="Malaga"/>	<input type="text" value="Andalucia"/>	<input type="text" value="29006"/>
<input type="button" value="Registrar Usuario"/>		

### Listado de Usuarios

Id	Name
1	Emerson Beard
2	Tabatha Matthews

Al pulsar el botón se genera un evento que recoge nuestro fichero *users.js* , en concreto llamamos a la función **registerUser()** , que se encargará primero de hacer una pequeña validación para que no venga ningún campo vacío y despues manda los datos del formulario a nuestro controlador que es *UsersController.php* tal y como vemos en la petición ajax

Ver en la captura de la siguiente página como generamos a través de JavaScript la petición ajax para enviarla al controlador. Este pequeño paso que damos de por medio entre la vista y el controlador es específico de la programación de PHP, ya que este lenguaje por sí solo (sin frameworks de por medio) no tiene la capacidad para responder a eventos generados por el explorador.

Resumiendo, se ha generado un evento desde la vista (en este caso mandar unos datos de registro al pulsar su botón) , procesamos este evento a través de JS para enviarlo al controlador, esta petición ajax quedará a la espera de procesar los datos que devolverá más adelante el controlador.



../public/js/users/users.js

```
30
31 var DOMStringsUser = {
32     inputName: 'inputName',
33     inputEmail: 'inputEmail',
34     inputNick: 'inputNick',
35     inputPassword: 'inputPassword',
36     inputAddress: 'inputAddress',
37     inputCity: 'inputCity',
38     inputState: 'inputState',
39     inputZipCode: 'inputZip'
40 };
41
42 function registerUser() {
43     if(!validateNewUser()){
44         swal({
45             text: 'No debe haber campos vacios...',
46             type: 'error',
47             confirmButtonText: 'Ok'
48         });
49         return false;
50     }
51
52     data = {serviceType: 'register_user'};
53     for (let key in DOMStringsUser) {
54         data[''+ key +''] = document.getElementById(`${DOMStringsUser[key]}`).value;
55     }
56
57     $.ajax({
58         type: 'POST',
59         url: '../app/controllers/users/UsersController.php',
60         data: data,
61         dataType: 'json',
62         success: function (r) {
63             swal({
64                 text: r.message,
65                 type: 'info',
66                 confirmButtonText: 'Ok'
67             });
68             if(r.success) {
69                 loadInitialUsers();
70                 cleanInputs();
71             }
72         }
73     });
74 }
```

2º.- En el **controlador** al que hemos llamado desde *users.js* => **UsersController.php** crearemos una instancia del modelo **UsersModel.php** para mandarle los datos que recibimos y que haga en este caso una consulta INSERT a la BD.

Es decir nuestro controlador está conectando los datos que hemos enviado desde la **vista** hasta el modelo de datos.

**../app/controllers/users/UsersController.php**

```
4
5  $params = (object) filter_input_array(INPUT_POST);
6  $request = new UsersController($params);
7
8  class UsersController {
9
10     private $db , $model , $params;
11
12     function __construct($params) {
13         $this->db = new Database();
14         $this->db->_connect();
15
16         $this->model = new UsersModel($this->db);
17         $this->params = $params;
18
19         $this->initRequest();
20     }
21
22     function initRequest() {
23         switch($this->params->serviceType) {
24             case 'load_users':
25                 $this->loadUsers();
26                 break;
27             case 'register_user':
28                 $this->registerUser();
29                 break;
30         }
31     }
32
33 }
```

../app/controllers/users/UsersController.php

```
42 function loadUsers() {
43     $result = $this->model->_getAll();
44     $res_arr = $this->db->processResult($result);
45
46     echo json_encode($res_arr);
47 }
48
49 function registerUser() {
50     $success = true;
51     $message = 'Usuario creado correctamente';
52
53     try {
54
55         $result = $this->model->_insNewUser($this->params);
56         if($result === false) { throw new Exception; }
57     }
```

3º.- En el **modelo** recibimos estos parámetros que nos manda el **controlador** para realizar la consulta correspondiente, en este caso un INSERT en la tabla de users con el nuevo usuario.

../app/models/users/UsersModel.php

```
3 class UsersModel {
4
5     private $table = "users";
6     private $fields = array(
7         'idUser', 'name', 'pass', 'nick', 'email', 'address', 'city', 'state', 'zipcode'
8     );
9
10    private $db_instance;
11    function __construct($db_instance = null) {
12        $this->db_instance = $db_instance;
13    }
14
15    function _getAll() {
16        if(!isset($this->db_instance)){ return null; }
17
18        $sql = "SELECT * FROM $this->table";
19        return $this->db_instance->getSQL($sql);
20    }
21
22    function _insNewUser($params) {
23        if(!isset($this->db_instance)){ return null; }
24
25        $sql = "INSERT INTO {$this->table} ( " .implode(',',$this->fields). " )";
26        $sql .= " VALUES(null , '{$params->inputName}' , '{$params->inputPassword}' , '{$params->inputNick}' , '{$params->inputEmail}' , ";
27        $sql .= " '{$params->inputAddress}' , '{$params->inputCity}' , '{$params->inputState}' , '{$params->inputZipCode}')";
28        return $this->db_instance->execSQL($sql);
29    }
30 }
```

y devolvemos el resultado al controlador , en este caso Mysql nos devuelve true o false según se haya podido realizar la consulta o no.

4º.- En el **controlador** ahora recibimos la respuesta del modelo que a su vez devolveremos a la petición ajax de users.js que envió los datos del formulario aquí.

../app/controllers/users/UsersController.php

```
48
49 function registerUser() {
50     $success = true;
51     $message = 'Usuario creado correctamente';
52
53     try {
54
55         $result = $this->model->_insNewUser($this->params);
56         if($result === false) { throw new Exception; }
57
58     } catch(Exception $ex){
59         $success = !$success;
60         $message = "Error al crear nuevo usuario. $ex->message";
61     }
62
63     $response = array('success' => $success , 'message' => $message);
64     echo json_encode($response);
65 }
66
67 }
68
```

5º.-Por último de nuevo en users.js procesaremos los datos devueltos por el controlador para si todo a ido bien mostrar la tabla con el nuevo registro o alertar con un mensaje de que no ha sido posible registrar el nuevo usuario. Es decir estaremos renderizando en la **vista** los datos nuevos que ha generado el **modelo** y que han sido procesados por el **controlador** que ha actuado de intermediario en todo este proceso.

../public/js/users/users.js

```
57 $.ajax({
58     type: 'POST',
59     url: '../app/controllers/users/UsersController.php',
60     data: data,
61     dataType: 'json',
62     success: function (r) {
63         swal({
64             text: r.message,
65             type: 'info',
66             confirmButtonText: 'Ok'
67         });
68         if(r.success) {
69             loadInitialUsers();
70             cleanInputs();
71         }
72     }
73 });
```

En la captura de abajo vemos como renderizamos en la **vista** los nuevos datos

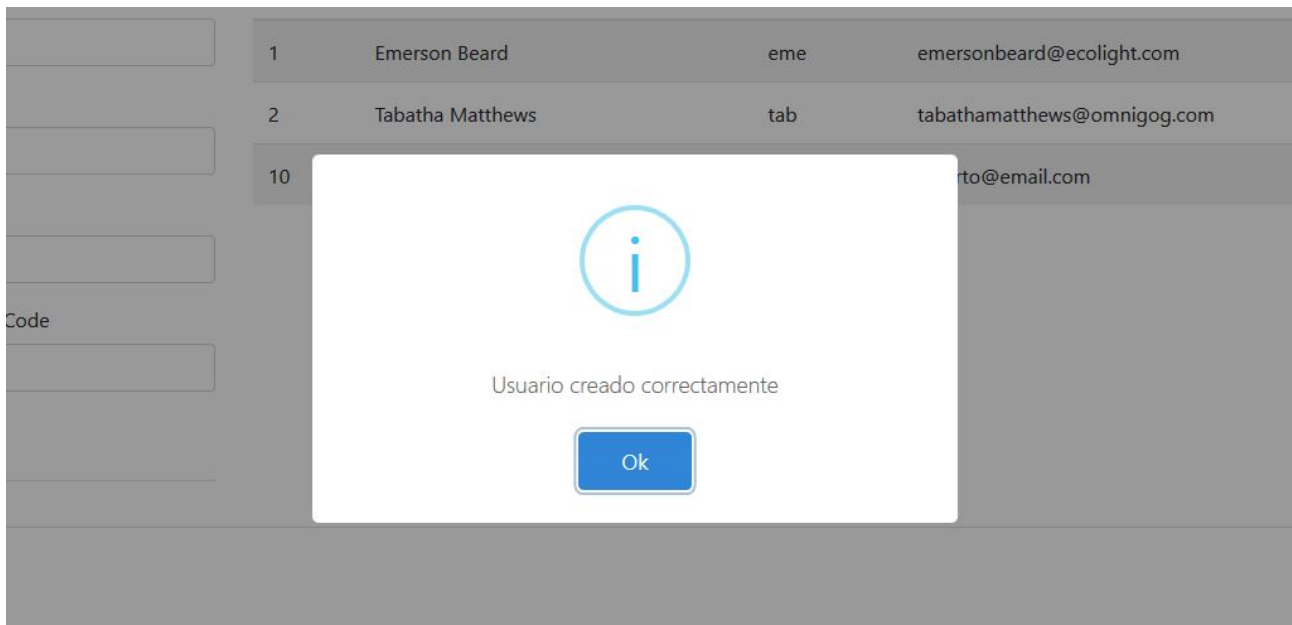
../public/js/users/users.js

```
19 function loadInitialUsers() {
20     $.ajax({
21         type: 'POST',
22         url: '../app/controllers/users/UsersController.php',
23         data: {serviceType: 'load_users'},
24         dataType: 'json',
25         success: function (r) {
26             renderTableUsers(r);
27         }
28     });
29 }
```

```
106 function renderTableUsers(resultSet) {
107     let html = '<table class="table table-striped mg-top-4">';
108     html += '<thead><tr>';
109     html += '<th scope="col">Id</th>';
110     html += '<th scope="col">Name</th>';
111     html += '<th scope="col">Nick</th>';
112     html += '<th scope="col">Email</th>';
113     html += '<th scope="col">State</th>';
114     html += '</tr></thead>';
115     html += '<tbody>';
116
117     for (let key in resultSet) {
118         let row = resultSet[key];
119         html += '<tr>';
120         html += '<td>${row.IDUser}</td>';
121         html += '<td>${row.name}</td>';
122         html += '<td>${row.nick}</td>';
123         html += '<td>${row.email}</td>';
124         html += '<td>${row.state}</td>';
125         html += '</tr>';
126     }
127
128     html += '</tbody></table>';
129     document.querySelector('#render-table-users').innerHTML = html;
130 }
```



finalmente , ya tendremos nuestra **vista** de nuevo preparada para seguir con la ejecución de la app



Listado de Usuarios					
					Buscar
	Id	Name	Nick	Email	State
	1	Emerson Beard	eme	emersonbeard@ecolight.com	New York
	2	Tabatha Matthews	tab	tabathamattthews@omnigog.com	Wisconsin
	10	Alberto	ATM	alberto@email.com	Andalucia