**SISTEMA PROVINCIAL MINORIDAD**

Como vamos a desarrollar?

El sistema será desarrollado con tecnologías open source de manera que el cliente pueda a futuro mantener el sistema por cuenta propia.

El sistema será desarrollado en PHP con soporte de framework cakephp en la versión mas reciente a la hora de iniciar el desarrollo.

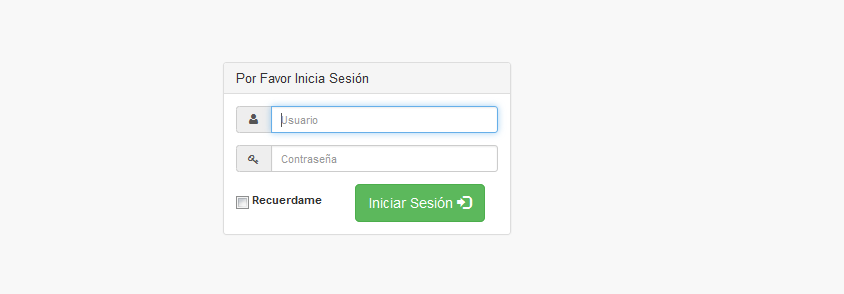
El motor de base de datos será Postgresql versión 9.1 u la versión mas reciente a la hora del comienzo del desarrollo.

**Interfaz.**

La interfaz será desarrollada basándose en un css de boostrao con los cambios que se crean oportunos.

Para la interacción del usuario se utilizara en su mayor parte jquery y jquery-ui para generar una interfaz amigable con el usuario.

Venta ingreso a sistema.



**Interfaz Administración**

La interfaz de administración general contará con

Una ventana index: se visualizan los datos del modelo que resultan relevantes para el usuario. La misma puede o no contener opciones de filtrado.

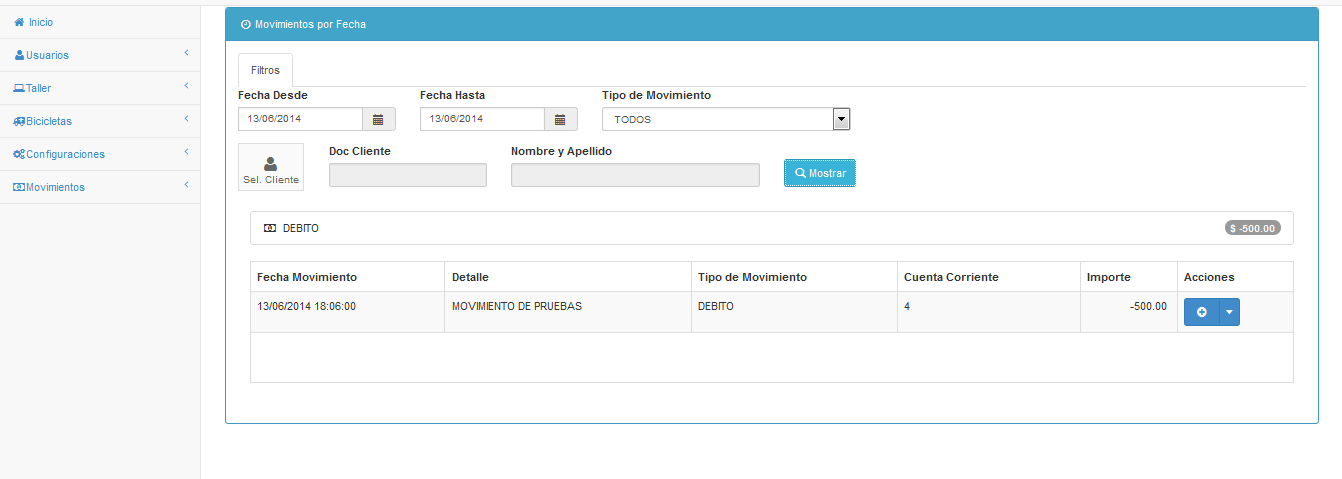
1: Nombre del modelo y detalle de la funcionalidad general.

2: Barra lateral de opciones de acceso.

3: Filtros que puedan requerirse con un botón para iniciar la búsqueda

4: Botón con menú desplegable para las operaciones posibles sobre el registro Borrar, Asignar,Imprimir etc

5: Paginado solo es visible cuando el objeto paginador encuentra datos





**Nota:** Todas las ventanas que llevan filtros, la carga de datos se realiza por medio de ajax de manera de no recargar la página de nuevo y para no perder los datos que el usuario esta filtrando.

Nota: Todas las ventanas index de listados deben llegar contenido el objeto toastmensaje y el código necesario para mostrar los mensajes generados con Session->SetFlash desde los controladores.

**Alta/Edición**

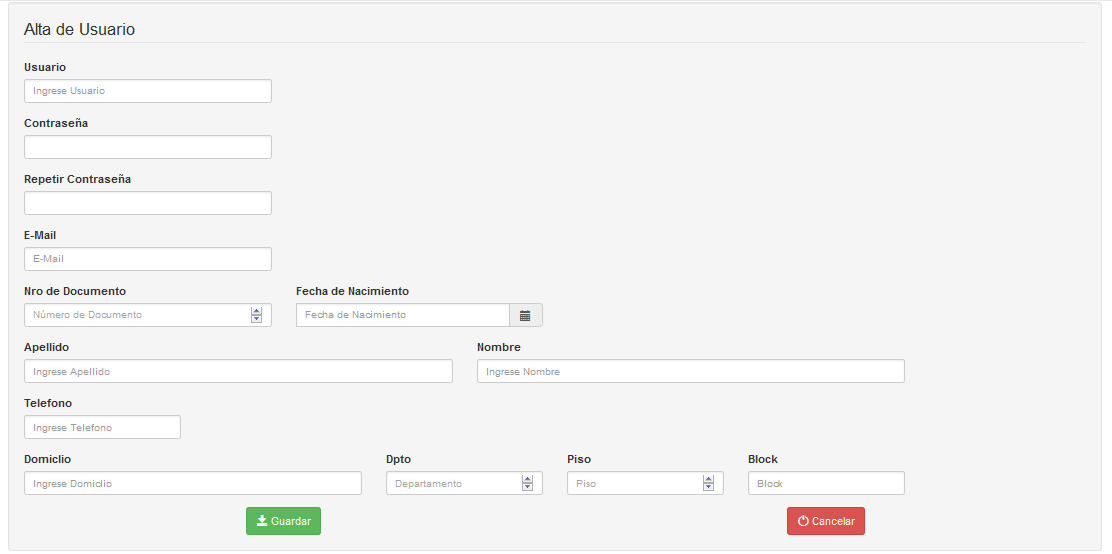
Permite realizar el alta u edición de un registro, el mismo puede ser una entidad solo u con otras entidades relacionadas

1: Especificación de la funcionalidad y luego ingresamos lo que estamos por realizar por ejemplo Nuevo Ingreso, Actualizar datos, Verificiar datos etc

2: Acciones a realizar sobre los datos. Grabar, Validar, Cancelar vuelve siempre a la ventana anterior index para el cancelar utilizar la funcionalidad de js.

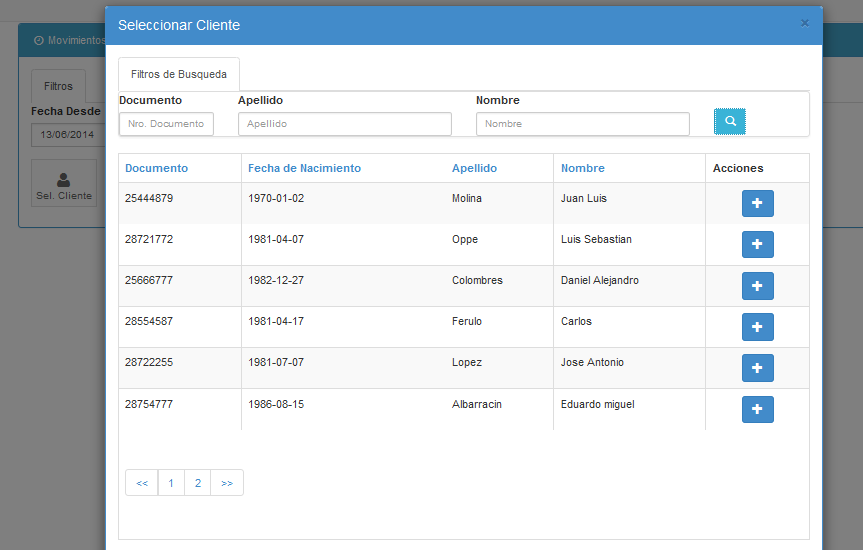
3: Datos a cargar del modelo, los errores de validación se visualizan siembre debajo de cada campo que dio error.

4: Las validaciones mas importantes, por ejemplo si tenemos un modelo y necesitamos conocer datos de otro modelo se realizan por medio de AJAX para no recargar la página. Solo ejecutamos las validaciones a nivel de servidor cuando los datos externos al modelo sean correctos u aquellos que podemos validar directamente en el modelo, ejemplo Fecha desde – Fecha hasta sean correctas etc.



**Selección de datos**

En el caso de requerir seleccionar datos de otro modelo se debe generar una opción para seleccionarlos por medio de una ventana modal para no perder de vista los datos que se encuentran cargando, por ejemplo si necesitamos seleccionar un cliente utilizamos una opción que muestre la ventana modal para seleccionar los datos y asignarlos al campo determinado.



**Se debe agregar el siguiente script cuando realizamos un llamado a un popup**

**$('#modalview').on('hidden.bs.modal', function () {**

**$('#content').empty();**

**$(this).data('bs.modal', null); //<---- empty() to clear the modal**

**})**

**Mascaras**

* Todos los campos en pantalla deben contener mascaras para cumplir con un formato u restricción, el campo DNI,CUIT deben tener las mascaras respectivas.
* Los campos numéricos no deben permitir ingresar letras.
* Los campos fecha se visualizan y modifican siempre con el formato dd/mm/yyyy
* La longitud de los campos deben ser validadas y tener un límite en el html.

**Programación.**

**Validaciones de Modelos**

Todas las validaciones de los modelos deben realizarse siempre sobre el modelo y de ser posible sobre el nivel de interfaz del cliente. Verificar y asignar a cada atributo la validación correspondiente para que cumpla con lo requerido en el modelo, verificar datos nulos, longitud, que sean del tipo establecido numéricos, email etc, cualquier validación propia del usuario debe ser creada en el modelo.

Las conversiones de datos u validaciones específica antes de guardar los datos deben ser realizadas en las funciones del beforeValida y beforeSave del framework. Ejemplo Convertir la fecha en un formato determinado, aplicar reemplazo de caracteres etc.

**Grabar datos en el modelo**

Para grabar datos de un modelo que contiene mas de una tabla relacionada se debe armar un proceso que valide y guarde los datos solo si en las n tablas se guardaron los datos caso contrario debe realizar un rollback de todos los datos y retornar el error, tener en cuenta las transacciones y su respectivo cierre y Rollback general.

**Variables Definiciones**

Las variables deben tener el **mismo nombre del** **modelo** en la vista para todos los casos, edición,alta, búsquedas etc. Ejemplo si tenemos un filtro por nombre de cliente entonces en el modelo para la creación del imput deberíamos crearlo con

$this->Form>input(**'Cliente.documento'**)

Si se pasa una estructura de datos la vista debe llevar el nombre del modelo principal en plural ejemplo clientes🡪estudios entonces al enviar los datos enviamos el array con el nombre <clientes>.

**JavaScript**

Las bibliotecas JS del modelo deben encontrarse en

/js/<nombre modelos en plural/bibliotecas.js siempre deben existir

Add.js

Edit.js

Index.js

**Código Comentado**

Encabezado para script php/javascript

/\*

\* @author <autor>

\* @Fecha <fecha creado>

\* @use <para que creamos el objeto u proceso breve descripción de funcionalidad>

\* @UsuadioModifica <usuario>

\* @FechaModificacion <fecha>

\* @DetalleModificacion <detalle>

\*

\*/

Funciones aisladas para php y JS

/\*

\* @use:<detalle de la función y utilización>

\*@params:

\* parametro1:<tipo detalle>

\* parametron:<tipo detalle>

\*/

**Stores Procedure**

Se realizará la utilización de procedimientos almacenados para aquellas operaciones que puedan resultar complejas u que requieran mucha programación a nivel de framwork u PHP, la utilización de stores procedures resulta practica y de mucha eficiencia al realizar cálculos u operaciones con grandes volúmenes de datos.

Un ejemplo es el cálculo de la edad de una persona basándonos en su fecha de nacimiento.

Esta sentencia puede ser generada y ejecutada con la orden find desde cakephp

Ejemplo Adjunto Sistema Demo:

SELECT documento,fechanac,s\_calcula\_edad(fechanac) AS

edad,apellido,nombre FROM clientes

Desde Cakephp desde el paginador

$this->paginate=array('limit' => 10,

'page' => 1,

'order'=>array('username'=>'desc'),

'conditions'=>$ls\_filtro,

'fields'=>array('documento','fechanac','s\_calcula\_edad(fechanac) AS edad','apellido','nombre'));

$this->Bicicleta->recursive = 0;

$this->set('bicicletas', $this->Paginator->paginate());

Si el proceso se puede resolver sin necesidad de utilizar SP hacerlo sobre el mismo SQL generado por el framework.

Otros casos de uso

* Procesamiento de lotes masivos
* Recuperación de datos de otros servidores u replicaciones.
* Cálculos complejos que puedan ser resueltos en forma eficiente en el mismo motor.

Desventajas:

El software queda pegado al motor de base de datos sin posibilidades de migrarlo a futuro.