# Diseño detallado

A continuacion definimos el diseño detallado partiendo de la arquitectura MVC, expandiendo cada componente de la arquitectura para proveer un mayor entendimiento de la aplicacion.

# Vista

Una vista es un script PHP que consiste basicamente en elementos de la interfaz de usuario.  
Basicamente la vista es la página HTML y el código que provee de datos dinámicos a la página.

Las vistas son las responsables de presentar los modelos en el formato que el usuario final desea.

Las vistas de la aplicacion estan divididas en dos grupos.

Vistas complejas: son las vistas que se componen de otras vistas y a su vez utilizan mas de un controlador y modelo.  
login, search, map

Vistas simples: de ABM (Alta Baja Modificación). Son las vistas que están asociadas a un modelo y un controlador y se utilizan para Obtener, Actualizar y Borrar el modelo asociado .

# Controlador

El controlador es el responsable de recibir los eventos de entrada desde la vista.

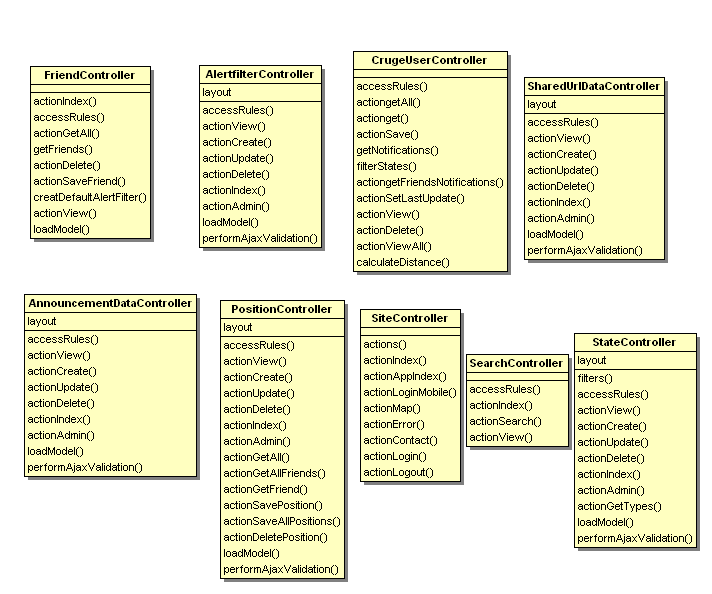
Cuando un controlador es invocado, ejecuta una acción que utiliza los modelos necesarios y muestra la información a travez de la vista apropiada.

Una acción, en su forma más simple, es un metodo de la clase controlador cuyo nombre comienza con action.

Los controladores y acciones están definidas por IDs. El ID del controlador se encuentra en la forma de path/to/xyz el cual es interpretado como el archivo de clase controlador protected/controllers/path/to/XyzController.php, donde xyz debe ser remplazada por el nombre de su controlador (ejemplo: post corresponde a protected/controllers/PostController.php). El ID de acción es el nombre del metodo sin el prefijo action. Por ejemplo si el controlador contiene el métodoactionEdit el ID de la acción correspondiente será edit.

Los usuarios realizan pedidos por un controlador y acción en términos de ruta. Una ruta se encuentra formada por la concatenación de un ID de controlador y un ID de acción separados por una barra. Por ejemplo la ruta post/edit se refiere a PostController y a su acción edit.

A continuacion se listan los controladores de la aplicacion junto con sus acciones.



**Explicacion Metodos**

Actions get, update, delete, save

Como se ve detallado, cada controlador define el CRUD( es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete).) de cada modelo. Se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software. Ademas le agregamos algunas funcionalidades como obtener o guardar todos los registros dados de un modelo (getAll, saveAll).

actionIndex: Se utiiza para listar todos los elementos de un modelo particular.  
  
actionAdmin: Metodo que se utiliza para definir un comportamiento particular si el usuario con privilegios de Administrador esta logueado.

loadModel: Se utiliza para cargar el modelo asociado al controlador.

performAjaxValidation: Posibilita realizar validación AJAX de un form sobre un modelo dado.

accessRules: Define los permisos de cada tipo de usuario a acciones del controlador.

Ejemplo controlador site

**public** **function** accessRules()

{

**return** **array**(

// allow all users to perform login action

**array**('allow',

'actions'=>**array**('login'),

'users'=>**array**('\*'),

),

// allow authenticated user to perform map action

**array**('allow',

'actions'=>**array**('map'),

'roles'=>**array**('@'),

),

// allow admin user to perform 'admin' action

**array**('allow',

'actions'=>**array**('admin'),

'users'=>**array**('admin'),

),

);

}

# Modelo

 Los modelos son utilizados para mantener los datos y sus reglas de negocio relevantes.

Un modelo representa un solo objeto de datos. El mismo hace referencia a una tabla de la base de datos.   
Cada campo del objeto de datos esta representado por un atributo en el modelo. El atributo tiene una etiqueta y esta se puede validar contra un juego de reglas.

A continuacion se listan todos los modelos de la aplicacion y su relacion entre ellos.



**Explicacion Metodos**

model: Constructor. Metodo que todos los modelos deben sobreescribir.

tableName: Devuelve el nombre de la tabla de la base de datos a la que representa el modelo.

Ejemplo del modelo Position

**public** **function** tableName()

{

**return** 'position';

}

rules:   Devuelve un arreglo con reglas de validacion definidas por el usuario. Estas se evaluan cada vez que se quiere utilizar un registro y es requerido que se cumplan para completar la accion.

Ejemplo del modelo State

**public** **function** rules()

{

**return** **array**(

**array**('id\_user, date, description', 'required'),

**array**('id\_user, id\_position', 'numerical', 'integerOnly'=>true),

**array**('type', 'length', 'max'=>50),

**array**('description', 'length', 'max'=>140),

// The following rule is used by search().

**array**('id, id\_user, date, type, description, id\_position', 'safe', 'on'=>'search'),

);

}

relations: Devuelve un arreglo donde están definido como se relaciona dicho modelo con otros.

Ejemplo del modelo State

**public** **function** relations()

{

**return** **array**(

**'announcementData' => array**(self::HAS\_ONE, 'AnnouncementData', 'id\_state'),

**'position'=> array**(self::BELONGS\_TO, 'Position', 'id\_position'),

**'idUser' => array**(self::BELONGS\_TO, 'CrugeUser', 'id\_user'),

);

}

attributeLabels : Cuando se diseña un form, generalmente se necesita mostrar una etiqueta para cada campo de entrada. La etiqueta le indica al usuario que clase de información se espera que cargue en el campo.   
Este método devuelve un arreglo que mapea cada atributo del modelo con una etiqueta definida por el usuario.

Ejemplo del modelo position

**public** **function** attributeLabels ()

{

**return** **array**(

**'latitude' =>** 'Latitude',

**'date' =>** 'Date',

**'id' =>** 'ID',

**'longitude' =>** 'Longitude',

);

}

search: Define los atributos que serán utilizados en los filtros de búsqueda.

**Herencia**

beforeSave: Metodo que es invocado antes de guardar un modelo. Generalmente utilizado con fines de validación.  
En este caso se uso para definir el atributo type del modelo State en la herencia.

instantiate: Metodo que crea una nueva instancia del modelo.   
Se utilizo para crear instancias del modelo State, cada vez que se creaba una instancia de uno de sus hijos.

defaultScope: Define el ámbito del modelo, el cual será utilizado en las búsquedas de la base de datos.  
Por ejemplo en nuestro caso: El defaultScope de announcement o sharedUrl es State.

Como se podra ver a continuacion en el diagrama de entidad relacion (DER) los modelos de la aplicacion estan directamente derivados de cada tabla de la base de datos.

Modelo de datos

# Descripcion Tablas:

State: Almacena la informacion comun a cualquier actualizacion de estado de un usuario de la red. Es una representacion del estado mas simple que solo posee una descripcion.  
  
Announcement\_data: Estado particular que extiende de State y agrega campos como una imagen y un mensaje.  
  
Shared\_url\_data: Estado particular que extiende de la table State y posee una url que permite compartir vinculos a otras paginas.  
  
Alertfilter: Almacena la distancia que define un usuario en base a su posicion con respecto a la posicion de otro usuario. Este valor sera usado como filtro para recibir las actualizaciones de estado del segundo usuario.

Position: Define posiciones mediante los valores de latitud y longitud. Adicionalmente posee una fecha en la cual se almaceno la informacion.

Cruge\_user: Almacena la informacion de cada usuario de la aplicacion. Posee informacion adcional relacionada con el modulo Cruge para el framework Yii para el manejo de usuarios y sesiones. (\*)

Friend: Define la relacion de Amistad entre dos usuarios registrados en la table cruge\_user. Adicionalmente posee una fecha en la cual se almaceno la informacion.

User\_position\_relation: Tabla que representa la relacion entre un usuario y las posiciones almacenadas.

User\_pic\_relation: Tabla que almacena las imagenes de perfil que puede tener un usuario.