

Cruge

De Yii Framework en Español (Yiiñ)

Por: **Christian Salazar**

Seguir a @salazarchris74

Contenido

- 1 TUTORIAL DE CRUGE
- 2 CAPTURAS DE PANTALLA DE CRUGE / SCREENSHOTS
- 3 INSTALACIÓN:
 - 3.1 LEE EL TUTORIAL
 - 3.2 PASO 1 DESCARGA
 - 3.3 PASO 2 ARCHIVO CONFIG/MAIN
 - 3.4 PASO 3 SCRIPT DE BASE DE DATOS
 - 3.5 PASO 4 (HELPER) MENU POR DEFECTO
 - 3.6 PASO 5. VER CONSOLA DE ERROR
 - 3.7 PASO 6. CrugeAccessControlFilter
 - 3.8 PASO 7. CONFIGURAR EL USUARIO INVITADO
 - 3.9 ERRORES FRECUENTES
- 4 CÓMO ACTUALIZAR TU ACTUAL INSTALACIÓN DE CRUGE
- 5 QUÉ ES CRUGE ?
 - 5.1 Conceptos Básicos de un sistema basado en Roles - formal
 - 5.2 Un ejemplo de uso de roles
- 6 EJEMPLOS DEL USO DEL API DE CRUGE:
 - 6.1 CONOCER EL ID DEL USUARIO ACTIVO
 - 6.2 CONOCER SI EL USUARIO ES INVITADO
 - 6.3 CONOCER SI EL USUARIO ES SUPER ADMINISTRADOR
 - 6.4 OBTENIENDO EL EMAIL DEL USUARIO ACTIVO
 - 6.5 VERIFICAR SI TIENE PERMISO:
 - 6.6 BUSCAR UN USUARIO POR SU ID
 - 6.7 BUSCAR UN USUARIO POR SU USERNAME
 - 6.8 BUSCAR UN USUARIO POR SU USERNAME o EMAIL
 - 6.9 BUSCAR UN USUARIO SEGUN EL VALOR DE UN CAMPO PERSONALIZADO
 - 6.10 VERIFICAR SI UN USUARIO CUALQUIERA HA INICIADO

SESION

- 6.11 LEYENDO "EL VALOR" DE UN CAMPO PERSONALIZADO DE UN USUARIO
- 6.12 LEYENDO "EL OBJETO CRUGEFIELDVALUE" DE UN CAMPO PERSONALIZADO DE UN USUARIO
- 6.13 LEER TODOS LOS CAMPOS PERSONALIZADOS DE UN USUARIO
- 6.14 RECORRIENDO VARIOS USUARIOS Y LEER SU EMAIL
- 6.15 LISTAR USUARIOS QUE TENGAN UN ROL ASIGNADO
- 6.16 LISTAR USUARIOS CON CAMPOS PERSONALIZADOS (ARRAY O DATAPROVIDER)
- 6.17 ARMANDO UN COMBOBOX CON LA LISTA DE USUARIOS REGISTRADOS MOSTRANDO LOS CAMPOS PERSONALIZADOS (EJEMPLO)
- 6.18 EDITAR UN CAMPO PERSONALIZADO (EJEMPLO)
- 6.19 GENERAR UN USERNAME AUTOMATICAMENTE BASADO EN EL EMAIL
- 6.20 CREAR NUEVO USUARIO (recomendado)
- 6.21 CREAR NUEVO USUARIO (otra forma)
- 6.22 CREAR USUARIOS FICTICIOS (ALEATORIOS)
- 6.23 LOGIN USER
- 6.24 REMOTE LOGIN
- 7 VARIABLES DEL SISTEMA
- 8 MANEJANDO ENLACES
- 9 USO DE LAYOUTS
- 10 CRUGE RBAC
 - 10.1 Usar checkAccess para verificar si el usuario tiene permiso
 - 10.2 Controlando el Acceso a un "controller" o a un "action" usando CrugeAccessControlFilter
 - 10.3 Saber por donde paso un usuario
 - 10.4 Programacion del RBAC: diferencias con otras extensiones.
 - 10.5 Modo de Programación del RBAC
 - 10.5.1 Desplegando una lista de permisos requeridos al pie de tu página
 - 10.5.2 Usando el LOG
 - 10.5.3 Tips de Programacion del RBAC
- 11 Superusuario e Invitados
 - 11.1 Seleccionando el modo de inicio de sesión: Por usuario, por Email, o ambos
 - 11.2 El superUsuario
 - 11.3 El usuario Invitado
 - 11.4 TRAS INSTALAR, DEBES CONFIGURAR EL USUARIO INVITADO

- 12 Manejo de Eventos con Cruge
 - 12.1 EVENTOS DE INICIO, CIERRE Y EXPIRACIÓN DE SESIÓN
 - 12.2 LOGIN, LOGOUT, SESIONES. CÓMO FUNCIONA EN CRUGE.
 - 12.3 Conviviendo con CAccessControl filter.
 - 12.4 Ciertos Actions Requieren Inicio de Sesion
 - 12.5 FILTROS
 - 12.6 CÓMO SE OTORGA UNA SESION EN CRUGE y COMO ESTA EXPIRA
 - 12.7 CONTROL AVANZADO DE SESIONES Y EVENTOS DE AUTENTICACION Y SESION
 - 12.7.1 ¿ Cómo Redirigir al usuario a una pagina especifica tras iniciar sesión ?
 - 12.8 CONTROL AVANZADO DE EVENTOS DE INSERCIÓN Y ACTUALIZACIÓN DE USUARIOS
- 13 EL ENVIO DE CORREOS CON CRUGEMAILER
 - 13.1 Configurar CrugeMailer
 - 13.2 Personalizar CrugeMailer
 - 13.3 ¿ Cómo cambiar el método de envío de correo ? (mail, phpmailer etc)
 - 13.4 Beneficios obtenidos al usar CrugeMailer
 - 13.5 ¿Cómo hacer un CRON (BATCH) con CrugeMailer ?
 - 13.6 CASO REAL: Crea tu propia clase MiCrugeMailer
- 14 ENCRIPTADO DE CLAVES
 - 14.1 CÓMO ACTIVAR LA ENCRIPTACION
 - 14.2 CÓMO SELECCIONAR EL METODO DE ENCRIPTACION
 - 14.3 COMO ENCRIPTAR LA CLAVE DE ADMIN
- 15 MANEJO DE ERRORES (EXCEPCIONES) EN CRUGE
- 16 ERRORES FRECUENTES
 - 16.1 Excepcion: no se pudo hallar el sistema de configuracion, quiza la tabla cruge_system esta vacia o ha indicado un identificador de sistema inexistente
 - 16.2 Excepción: por favor cambie las referencias a 'useridentity' por 'crugeuser'
 - 16.3 Alias "application.modules.cruge.components.CrugeWebUser" is invalid. Make sure it points to an existing PHP file and the file is readable.
 - 16.4 The authenticity of host 'bitbucket.org (207.223.240.182)' can't be established.
 - 16.5 Esta página web tiene un bucle de redireccionamiento. Error 310 (net::ERR_TOO_MANY_REDIRECTS): Demasiados redireccionamientos
- 17 I18N
- 18 CASOS DE EJEMPLO

- 18.1 Cruge con Fancybox y Ajax
- 18.2 Tu propia vista de edicion de perfil de usuario
- 18.3 Acceder a los campos Personalizados desde tu propia clase relacionada con CrugeUser
- 18.4 Presentar campos personalizados de CrugeUser en un CGridView
- 18.5 El empleado debe pertenecer a una o varias empresas
- 18.6 Mostrando Campos Personalizados en una Relacion
- 18.7 Personalizar la lista de valores de opcion de un Campo Personalizado tipo LISTBOX
- 19 CREANDO MENU ITEMS BASADOS EN RBAC CON CRUGE
- 20 CRUGE y BOOTSTRAP
- 21 CONECTAR CRUGE CON FACEBOOK Y GOOGLE
- 22 DIAGRAMAS UML DE CRUGE
 - 22.1 Diagrama de Clases / Secuencia de Cruge y MiSesionCruge
 - 22.2 Diagrama de Clases Cruge
 - 22.3 Diagrama Actividad (caso "login")
 - 22.4 Diagrama de Secuencia (caso "login")
 - 22.5 Diagrama de Clases de Yii Auth Base

TUTORIAL DE CRUGE

Aqui puedes leer un Tutorial de Cruge.

CAPTURAS DE PANTALLA DE CRUGE / SCREENSHOTS

| | | | |
|---|---|--|---|
|  |  |  |  |
| inicio de sesión | (pantalla de registro, que incorpora las opciones seleccionadas | (Pantalla para recuperar la contraseña, protegida | (Pantalla de administración de usuarios) |

por el administrador, incluyendo campos personalizados, captcha, terminos etc. Todo es opcional excepto usuario y clave.)

por captcha. Envío de clave al correo. Usuario recibe una nueva clave encriptada o la clave literal según las preferencias de encriptado.)

menu it

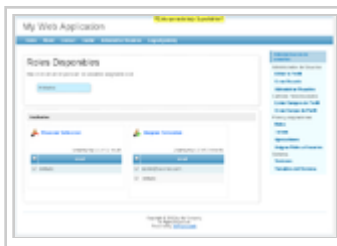


(Opciones del sistema. Aquí se configura el comportamiento de la pantalla de registro, las sesiones, y otras opciones como por ejemplo detener el servicio etc.)

(pantalla de edición del perfil del usuario activo o del usuario seleccionado (si es un administrador). Se pueden apreciar los campos personalizados.)

(pantalla de edición de los campos personalizados, con opciones variadas de validación, regexp etc.)

(pantalla de roles, totalmente operativa por clic y a o tarea seleccionada)



(pantalla de asignación de roles a los usuarios seleccionados.)

INSTALACIÓN:

LEE EL TUTORIAL

Aqui puedes leer un Tutorial de Cruge.

PASO 1 DESCARGA

Principalmente necesitas obtener cruge, puedes hacerlo por dos vías:

- Via descarga directa desde Repositorio oficial de Cruge (<https://bitbucket.org/christiansalazarh/cruge>) , busca el enlace "download" en la ventana de la derecha junto a "Size (1.5MB)...(Download)". Simplemente descomprimirás el paquete cruge.zip obtenido dentro de tu proyecto asi:

```
/home/tuapp/protected/modules/cruge
```

si tu directorio modules no existe, deberás crearlo a mano anteriormente (obviamente).

- Vía GIT. Asumo que has configurado tu cliente GIT y que funciona bien. Personalmente recomiendo esta vía, es mas seria y permite que tu proyecto se mantenga al día con el repositorio oficial. Si obtienes cruge por via GIT, haces lo siguiente:

```
cd /home/tuusuario/tuapp/protected
mkdir modules
cd modules
git clone https://christiansalazarh@bitbucket.org/christiansalazarh/cruge.git
# si usas git con ssh puedes usar:
# git clone git@bitbucket.org:christiansalazarh/cruge.git
```

NOTA, SI USAS GIT: Si no tienes una clave SSH creada, deberás usar HTTP en vez de SSH, mira este issue para mayor información: [issue#24](https://bitbucket.org/christiansalazarh/cruge/issue/24/error-al-hacer-clone-del-repositorio-cruge) (<https://bitbucket.org/christiansalazarh/cruge/issue/24/error-al-hacer-clone-del-repositorio-cruge>)

PASO 2 ARCHIVO CONFIG/MAIN

Ahora, a configurar Cruge, editarás el siguiente archivo:

```
/home/tuusuario/tuapp/protected/config/main.php
```

```

1. dentro de 'import' agregar:
    'application.modules.cruge.components.*',
    'application.modules.cruge.extensions.crugemailer.*',

2. dentro de 'modules' agregar:
    'cruge'=>array(
        'tableprefix'=>'cruge_',

        // para que utilice a protected.modules.cruge.models.auth.CrugaAuthDefau
        //
        // en vez de 'default' pon 'authdemo' para que utilice el demo de autent
        // para saber mas lee documentacion de la clase modules/cruge/models/auth
        //
        'availableAuthMethods'=>array('default'),

        'availableAuthModes'=>array('username','email'),

        // url base para los links de activacion de cuenta de usuario
        'baseUrl'=>'http://coco.com/',

        // NO OLVIDES PONER EN FALSE TRAS INSTALAR
        'debug'=>true,
        'rbacSetupEnabled'=>true,
        'allowUserAlways'=>true,

        // MIENTRAS INSTALAS..PONLO EN: false
        // lee mas abajo respecto a 'Encriptando las claves'
        //
        'useEncryptedPassword' => false,

        // Algoritmo de la función hash que deseas usar
        // Los valores admitidos están en: http://www.php.net/manual/en/function
        'hash' => 'md5',

        // Estos tres atributos controlan la redirección del usuario. Solo serán
        // hay un filtro de sesion definido (el componente MiSesionCruga), es me
        // lee en la wiki acerca de:
        // "CONTROL AVANZADO DE SESIONES Y EVENTOS DE AUTENTICACION Y SESION"
        //
        // ejemplo:
        //
        //         'afterLoginUrl'=>array('/site/welcome'), ( !!! no olvid
        //         'afterLogoutUrl'=>array('/site/page','view'=>'about'),
        //
        'afterLoginUrl'=>null,
        'afterLogoutUrl'=>null,
        'afterSessionExpiredUrl'=>null,

        // manejo del layout con cruge.
        //
        'loginLayout'=>'//layouts/column2',
        'registrationLayout'=>'//layouts/column2',
        'activateAccountLayout'=>'//layouts/column2',
        'editProfileLayout'=>'//layouts/column2',
        // en la siguiente puedes especificar el valor "ui" o "column2" para que
        // de fabrica, es basico pero funcional. si pones otro valor considera
        // requerirá de un portlet para desplegar un menu con las opciones de ad
        //
        'generalUserManagementLayout'=>'ui',

        // permite indicar un array con los nombres de campos personalizados,
        // incluyendo username y/o email para personalizar la respuesta de una c
        // $usuario->getUserDescription();
        'userDescriptionFieldsArray'=>array('email'),

    ),

```

```

3.    dentro de 'components' agregar:
      //
      //  IMPORTANTE: asegurate de que la entrada 'user' (y format) que por defecto t
      //                sea sustituida por estas a continuación:
      //
      'user'=>array(
          'allowAutoLogin'=>true,
          'class' => 'application.modules.cruge.components.CrugeWebUser',
          'loginUrl' => array('/cruge/ui/login'),
      ),
      'authManager' => array(
          'class' => 'application.modules.cruge.components.CrugeAuthManager',
      ),
      'crugemailer'=>array(
          'class' => 'application.modules.cruge.components.CrugeMailer',
          'mailfrom' => 'email-desde-donde-quieres-enviar-los-mensajes@xxxx.com',
          'subjectprefix' => 'Tu Encabezado del asunto - ',
          'debug' => true,
      ),
      'format' => array(
          'datetimeFormat'=>"d M, Y h:m:s a",
      ),

```

PASO 3 SCRIPT DE BASE DE DATOS

Ahora, debes preparar la base de datos, crearás las tablas requeridas en la base de datos de tu aplicacion, usa el script de acuerdo a tu motor de datos: mysql o postgre. Por favor anota que **tu usuario por defecto será admin y su clave admin**.

Usa el script SQL que cruge provee, copia y pégalo en tu mysqladmin o simplemente importalo, usa la vía que mejor te funcione, a veces importar no funciona en cambio copiar y pegar si.

```
/home/tuusuario/tuapp/protected/modules/cruge/data/cruge-data-model.sql (script de mysql)
```

PASO 4 (HELPER) MENU POR DEFECTO

Configura el menu de tu aplicación para que incorpore un acceso al item de "administración de usuarios" de Cruge, para ello editas tu archivo:

```
/home/tuusuario/tuapp/protected/views/layouts/main.php
```

en el sustituyes el componente CMenu por el que te doy a continuacion.

```
<?php $this->widget('zii.widgets.CMenu',array(
    'items'=>array(
```



```

        array('label'=>'Home', 'url'=>array('/site/index')),
        array('label'=>'About', 'url'=>array('/site/page', 'view'=>'about')),
        array('label'=>'Contact', 'url'=>array('/site/contact')),
        array('label'=>'Administrar Usuarios'
            , 'url'=>Yii::app()->user->ui->userManagementAdminUrl
            , 'visible'=>!Yii::app()->user->isGuest),
        array('label'=>'Login'
            , 'url'=>Yii::app()->user->ui->loginUrl
            , 'visible'=>Yii::app()->user->isGuest),
        array('label'=>'Logout ('.Yii::app()->user->name.)'
            , 'url'=>Yii::app()->user->ui->logoutUrl
            , 'visible'=>!Yii::app()->user->isGuest),
    ),
); ?>

```

Nota aqui: Es posible que puedas usar el formulario por defecto de Yii (views/login.php), pero para hacerlo funcionar con Cruge debes cambiar en models/FormLogin.php las dos lineas que dicen UserIdentity por CrugeUser. Ver mas detalles (<https://bitbucket.org/christiansalazarh/cruge/issue/16/excepcion-por-favor-cambie-las-referencias>)

PASO 5. VER CONSOLA DE ERROR

Casi terminado. Ahora Edita tu archivo "protected/views/layouts/main.php" y pon esta linea antes de la linea </body> es para que puedas visualizar que permisos se van requiriendo en tu aplicación a medida que el usuario las va usando (valga la redundancia):

```
<?php echo Yii::app()->user->ui->displayErrorConsole(); ?> Más información aquí
```

PASO 6. CrugeAccessControlFilter

En cada Controladora a la cual quieras controlar el acceso usando Cruge deberás seguir estos pasos

PASO 7. CONFIGURAR EL USUARIO INVITADO

Para finalizar es importante que configures al usuario Invitado: Lee mas

ERRORES FRECUENTES

Mira una lista de errores frecuentes.

Cualquier error reportalo aqui: Issues de Cruge (<https://bitbucket.org/christiansalazarh/cruge/issues/new>) .

CÓMO ACTUALIZAR TU ACTUAL INSTALACIÓN DE CRUGE

Bueno, en primer lugar si yo tuviese una manera de "prohibir" que cruge se instalara haciendo copy-paste del zip download, lo haría, la razón es la existencia de GIT:

- Con dos comandos de GIT puedes actualizar toda tu instalacion de Cruge.
- Cuando no usas GIT, te toca actualizar a mano.

CON GIT:

```
1- abres una consola.  
2- cd /blabla/miaplicacionweb/protected/modules/cruge  
3- git fetch  
4- git pull
```

LISTO. con eso tu instalación de Cruge habrá recibido los cambios desde el repositorio oficial.

ESO SI: Si le metiste mano al core de cruge (a cualquiera de sus archivos)..tendrás problemas. Por esta razon y otras pido que no hagan cambios al core de Cruge, ya que Cruge de por si no necesita de esos cambios, puedes usarlo y adaptarlo -sin tocarlo-. (y si hay algo que Cruge no tiene y necesito arreglarlo ?, pues bien, primero seria bueno que envíes ese requisito acá, para ver como te ayudamos, y segundo si no hay manera pues bien trata en lo posible que el cambio pueda hacerse de forma externa, por ejemplo creando una clase extendida, si aun no hay manera bueno deberás saber que al actualizarte esos cambios tendrán conflicto.)

QUÉ ES CRUGE ?

Aqui puedes leer un Tutorial de Cruge.

Cruge es un Framework para gestión de Usuarios y RBAC para Yii Framework. Te permite administrar y controlar de forma muy eficiente y segura a tus usuarios y los roles que ellos deban tener en tu Aplicación Web usando tanto un API Visual prefabricada y poniendo a tu disposición un API para controlar usuarios, login, sesiones a nivel de código evitando que tengas que acceder a estas partes empezando de cero.

¿ Por qué se llama Cruge ?

Control (de) Roles, Usuarios y Grupos (Extensión). El nombre Cruge se lo debemos a Esteban Perez (ver contribuciones al inicio).

¿ Y que pasó con la G de Grupos ?

Paciencia. A futuro, se incorporará un mecanismo de organización de usuarios en Grupos. Si bien los roles cumplen ese papel también es cierto que los Grupos ayudan a organizar usuarios para aplicar sobre ellos Roles u otras cosas. Por tanto..podríamos llamarlo: Cru_e (sin la g, pero suena feo). :-)

Conceptos Básicos de un sistema basado en Roles - formal

Los niveles de protección básicos de un sistema RBAC como Cruge son: ROLES, TAREAS y OPERACIONES, las cuales no son el gran descubrimiento del equipo que diseño a Yii Framework, en cambio son una teoría formal bien conocida que ellos implementaron para Yii Framework de una forma básica pero útil y bien hecha.

Un sistema RBAC como lo es Cruge, esta organizado de modo que un administrador principal pueda controlar a sus usuarios. La teoría formal (y es bueno que la comprendas) es que esto no es un invento de la rueda, un sistema RBAC tiene sus fundamentos y fuentes teóricas formales duras, aquí puedes ver una de esas fuentes formales acerca de los sistemas RBAC (http://es.wikipedia.org/wiki/Identity_Management) .

Es bueno comprender por qué esta organizado así (roles<-tareas<-operaciones) y lo explico con un ejemplo práctico:

La empresa JAMBOREE tiene un sistema administrativo, el cual tiene varios procesos delicados, concentremonos en el proceso de "emisión de cheques". Un administrador puede asignar a Juan al ROL "EMISIONCHEQUES", pero a su vez este ROL puede requerir de varias TAREAS que quiza esten en uso en otros ROLES también, y a su vez, cada TAREA esta compuesta de OPERACIONES, las cuales son indivisibles (o atómicas, una operacion -no debe- estar compuesta de operaciones por razones de buena práctica) además cada tarea de éstas pudiera estar comprometida en otras tareas de otros roles. Así de complejo puede ser, por eso *es necesario ser ordenado y objetivo al momento de diseñar el nivel de permisos o roles de un sistema RBAC*. A esto lo llamo "Anidación". Sigue leyendo.

Siguiendo este ejemplo (párrafo de arriba), para emitir un cheque se necesita que un usuario invitado pueda hacer login, pueda "acceder_al_menu_principal" y

pueda "listar_los_cheques_emitidos", para finalmente ejecutar el formulario "crearcheque". Pero a la vez, podríamos tener a otro ROL, aquel del usuario supervisor de cheques emitidos, el cual puede hacer lo mismo a excepción de "crearcheque". Aquí verás a que me refiero con el concepto de "Anidación".

Por tanto, cómo diseñamos ese complejo sistema de permisos ? Con un montón de operaciones todas asignadas a un solo rol ? lo haz hecho así ? pues esta mal hecho, y es grave desordenado y erróneo. Se hace así:

ROL_EMISIONCHEQUES

- compuesto de:
 - TAREA_acceder_al_menu_principal
 - TAREA_listar_los_cheques_emitidos
 - TAREA_crearcheque

ROL_SUPERVISORCHEQUES

- compuesto de:
 - TAREA_acceder_al_menu_principal
 - TAREA_listar_los_cheques_emitidos

Como ves, hay tareas en común, además de que la tarea "crearcheque" no esta asignada al rol del supervisor.

¿ Y dónde están las OPERACIONES ? Te habrás preguntado (si me vas siguiendo con atención)...pues las operaciones son el detalle menor, son todos aquellos pasos menores que tu puedes poner en tu sistema regados como arroz por todas partes, las cuales pueden ser en el caso de yii framework los "actions" que comprenden a tu aplicación. Tu asignas a tus usuarios ROLES y TAREAS...y las operaciones déjalas en los detalles propios de cada tarea.

Así continuando, las operaciones de TAREA_acceder_al_menu_principal pudieran ser: action_site_index, action_site_menusprincipal y quizá una que otra operación flotante que tu verificas manualmente mediante el api de Cruge:

```
Yii::app()->user->checkAccess('solo_usuarios_registrados');
```

Ahora, Yii (o mejor dicho la funcion checkAccess creada por Maurizio Domba de Yii Core) es lo bastante bien hecha como para comprobar la "anidación", supón que en algún lado se hace esta verificación:

```
if(Yii::app()->user->checkAccess("action_site_index")) {...}
```

Ahora supón además que el usuario activo SOLO tiene asignado el rol "ROL_EMISIONCHEQUES" y nada mas. Pues bien, el API y la función checkAccess harán una búsqueda con anidación (recursiva) para ver si el usuario

activo tiene ese item solicitado: "action_site_index", así sea directamente o mediante la asignacion de esta a una tarea que a su vez esta incluida en un ROL.

En Cruge (y en Yii en general) los items: ROLES, TAREAS y OPERACIONES son internamente lo mismo, ojo sin confusiones aquí, internamente son lo mismo pero para el sistema, no son lo mismo conceptualmente. Son niveles con jerarquía, las tareas no se asignan a operaciones (no se debe, aunque yii lo permita).

Arquitectura Interna de Cruge

Cruge tiene una alta Arquitectura OOP, basada en interfaces, lo que ayuda enormemente a usarla sin modificar en lo absoluto su propio core. Si necesitas cambiar de ORDBM, cruge lo permite. Si necesitas extender el funcionamiento de autenticacion para admitir nuevos metodos tambien lo permite mediante la implantacion de filtros de autenticacion, incluso dispones ademas de filtros insertables para controlar el otorgamiento de una sesion a un usuario y finalmente para controlar los registros y actualizaciones de perfil de tus usuarios. Todo eso sin tocar en lo absoluto el core de Cruge.

Cruge es un API, que incluye una interfaz de usuario predeterminada con el objeto que puedas usar el sistema como viene ahorrandote mucho tiempo. Esta interfaz hace uso del API de forma estricta, es decir, no hay "dependencias espagueti" las cuales son las primeras destructoras de todo software.

Listado de API's disponibles en Cruge:

Yii::app()->user

Para acceder a funciones muy básicas del control de usuarios. Compatible al 100% con Yii Estándar.

Yii::app()->user->ui

Provee enlaces a funciones de la interfaz

Yii::app()->user->um

Provee acceso al control de los usuarios

Yii::app()->user->rbac

Provee acceso a las funciones de RBAC.

Organización interna de Cruge - Sin dependencias Espagueti!!

[Tu Aplicacion]--->[Cruge]--->[API]---->[Factory]---->[modelos]

Esto significa que aún dentro de Cruge las dependencias son estrictamente organizadas, es decir, no verás interdependencias, si eso fuera así estaríamos

hablando de otra "extension" espagueti, como aquellas que solo le funcionan a su creador, o que en el mejor de los casos, funcionan...pero manipulandoles el core!!

Explayando mas este punto, este pequeño diagrama (arriba) te muestra que, por ejemplo, el nivel API (supongamos `Yii::app()->user->um` o `Yii::app()->user->ui` o cualquiera de las de arriba mencionadas), no accede a otros niveles inferiores sin pasar por el nivel subyacente, dicho de otro modo, cada nivel depende del nivel inferior, esto ayuda a controlar las dependencias, ayudando a futuros colaboradores a entender mejor el código.

¿ Por qué hay un Factory ?

Por una sencilla razón: Un Factory es un punto intermedio al cual un API pregunta o pide algo y el Factory responde, evitando que el API acceda directamente a los modelos específicos, nuevamente ayudando a la organización del código.

Pero bueno...esto ya es parte de un tema acerca de patrones de diseño y demás asuntos que se salen del foco: El uso de Cruge. Ya vamos para allá.

Un ejemplo de uso de roles

Supongamos que tienes un sistema de una librería, en donde se te ofrece en tu cuenta listar "tus libros favoritos", por tanto podrás borrar, listar, crear, editar cada entrada de esa lista de "Tus Favoritos". Ese mismo sistema también tiene una gestión administrativa a la cual tu no debes entrar porque como habrás adivinado son funciones del dueño de la librería a la cual tu no tendrás acceso.

Basado en ese ejemplo, supongamos que tu te la das de jaker..y quieres entrar con tu cuenta, capturar la URL del enlace:

```
"<a href='index.php?r=tucuenta/eliminarFavorito&id=123'>Eliminar tu Favorito Seleccionado</a>"
```

y con eso pues...haces un desastre, cambiando el ID 123 "de tu favorito", por el ID 99128 el cual pudiera ser perfectamente el ID de un favorito de otro usuario....grave.

Pues no te sorprendas, hay sistemas que permiten eso, y luego le echan la culpa al jaker...no señor, eso es culpa de quien hizo el sistema: Por qué razón tu puedes ejecutar ese acción con un ID de una entrada de favoritos que no es tuya ??!.

En este caso de ejemplo es donde entran en juego dos cosas: El control de acceso RBAC, y el control de acceso de reglas de negocio. En ese bobo ejemplo de arriba solo se controla la eliminación de favoritos usando solo RBAC, pero debido a la

ausencia de validación de reglas de negocio, pues bien te han "jakeado".

Cómo lo resuelves.

En primer lugar, ese action "eliminarFavorito" que pertenece al controller "tucuenta" quien debe tener control de acceso por RBAC, sencillamente para que no venga un invitado a copiar la URL y a pegarla en su browser para tratar de hacerte un desastre en tu aplicación. Un caso mas elegante sería evitar que un usuario tenga acceso a los actions del dueño de la librería de ejemplo de este caso.

Por tanto, si el usuario jperez el cual es un usuario "raso" de la librería trata de acceder a otro lado, digamos a "index.php?r=admin/registrarlibro", pues recibirá una excepción de "access denied". Si por el caso "happy day" ese usuario raso jperez quiere acceder a su lista de favoritos podrá hacerlo.

Entonces, al programar -solo- el RBAC lograríamos evitar que alguna persona entre a alguna parte del sistema.

Pero ahora, que sucede con ese caso especial en donde un usuario con acceso a favoritos quiere eliminar los favoritos de otro usuario ? El sistema RBAC no ha fallado..le dio acceso al action "eliminarFavorito", porque en la programación de RBAC se le dijo que el usuario raso puede usarla, pero, para eliminar "SU" favorito, no el del vecino. Cómo logramos esto ?

Pues bien, incorporando lógica de negocio en ese action.

Cuidado. Yii y el mecanismo por defecto de AuthManager provee una manera para pasarle al sistema RBAC un argumento para que sea evaluado junto con el RBAC, esto para garantizar de alguna manera que solo el usuario dueño del favorito pueda eliminar solo su favorito. Esta funcion de AuthManager es útil, pero termina hipercomplicando los sistemas, por tanto -a mi- no me gusta, prefiero aplicar una de las reglas de negocio de Python que bien dice: "...mejor explícito, que implícito..".

Cómo protegemos ese action. Simplemente con algo como:

```
public function actionEliminarFavorito($idfavorito){
    $model = Favorito::model()->findByPk($idfavorito);
    if($model->idUsuarioCreador == Yii::app()->user->id){
        $model->delete();
    }else{
        throw new CHttpException(500,"Disculpe usted no puede eliminar el favorito de otro usuario.");
    }
}
```

No te sorprenderá saber o redescubrir que éste es el mecanismo que el generador

de Yii (Gii) crea para el metodo actionDelete al momento de hacer un CRUD.

EJEMPLOS DEL USO DEL API DE CRUGE:

Primero, para usar el API de usuarios de Cruge debes acceder por:

```
Yii::app()->user->um
```

a diferencia de solo "Yii::app()->user" la cual es el API estándar de funciones de Yii, Yii::app()->user->um te provee funciones de negocio mas altas.

CONOCER EL ID DEL USUARIO ACTIVO

```
$id = Yii::app()->user->id; // el estandar de Yii
```

CONOCER SI EL USUARIO ES INVITADO

```
$booleanResult = Yii::app()->user->isGuest; // el estandar de Yii
```

CONOCER SI EL USUARIO ES SUPER ADMINISTRADOR

```
$booleanResult = Yii::app()->user->isSuperAdmin; // no es estandar de yii, es de cruge.
```

OBTENIENDO EL EMAIL DEL USUARIO ACTIVO

```
$email = Yii::app()->user->email;
```

VERIFICAR SI TIENE PERMISO:

```
if(Yii::app()->user->checkAccess('xxx')) {...} // "xxx" es un nombre de rol, tarea u operacion
```

BUSCAR UN USUARIO POR SU ID

```
$usuario = Yii::app()->user->um->loadUserById(123 /*, true (para que cargue sus campos)*/);  
echo $usuario->username;
```


BUSCAR UN USUARIO POR SU USERNAME

```
// busca estrictamente por su username, a diferencia de loadUser() quien busca por email o username
$usuario = Yii::app()->user->um->loadUserByUsername('jperez' /*, true (para que cargue sus campos)*/);
```

BUSCAR UN USUARIO POR SU USERNAME o EMAIL

```
$usuario = Yii::app()->user->um->loadUser('admin@gmail.com',true);
echo $usuario->username;
true: es para indicar que cargue los valores de los campos personalizados. por defecto es : false.
```

BUSCAR UN USUARIO SEGUN EL VALOR DE UN CAMPO PERSONALIZADO

```
$usuario = Yii::app()->user->um->loadUserByCustomField('cedula', $cedula); /
```

leer la nota de este commit (<https://bitbucket.org/christiansalazarh/cruge/commits/f0a6268324f7ded079ab629359204ab98104e16f>)

VERIFICAR SI UN USUARIO CUALQUIERA HA INICIADO SESION

```
$anaMaria = Yii::app()->user->um->loadUser('ana.maria@gmail.com');
$_s = Yii::app()->user->um->findSession($anaMaria);
if($_s == null)
    echo "Ana Maria no ha iniciado sesion.";
```

LEYENDO "EL VALOR" DE UN CAMPO PERSONALIZADO DE UN USUARIO

```
// getFieldValue acepta un ID numerico de userid, o una instancia de CrugeStoredUser,
// en este ejemplo se usa el ID del usuario que inicio sesion pero pudo haberse pasado
// como argumento a "$usuario" hallado con: $usuario = Yii::app()->user->um->loadUserById(123,true);
$nombre = Yii::app()->user->um->getFieldValue(Yii::app()->user->id,'nombre');
echo "nombre=".$nombre
```

leer la nota de este commit (<https://bitbucket.org/christiansalazarh/cruge/commits/f0a6268324f7ded079ab629359204ab98104e16f>)

LEYENDO "EL OBJETO CRUGEFIELDVALUE" DE UN CAMPO PERSONALIZADO DE UN USUARIO

```
$nombre = Yii::app()->user->um->getFieldValueInstance(Yii::app()->user->id,'nombre'); // getFieldValue
echo "nombre=".$nombre->value;
// la diferencia es que se retorna al objeto CrugeFieldValue, pudiendose hacer: $nombre->save() y simi
```

LEER TODOS LOS CAMPOS PERSONALIZADOS DE UN USUARIO

aqui se hace uso de `CrugeStoredUser->getFields()` (<https://bitbucket.org/christiansalazarh/cruge/src/75175ecf5d05/models/data/CrugeStoredUser.php?at=master#cl-64>) para que liste todos los objetos de clase `CrugeField` (<https://bitbucket.org/christiansalazarh/cruge/src/75175ecf5d05/models/data/CrugeField.php?at=master>), notar ademas que el valor se lee mediante `"campo->fieldvalue"` debido a que se esta obteniendo el valor desde el objeto `CrugeField->getFieldValue()`. ver codigo (<https://bitbucket.org/christiansalazarh/cruge/src/75175ecf5d05/models/data/CrugeField.php?at=master#cl-85>)

```
foreach($usuario->fields as $campo)
    echo "<p>campo: ".$campo->longname." es: ".$campo->fieldvalue;"</p>";
```

RECORRIENDO VARIOS USUARIOS Y LEER SU EMAIL

```
foreach(Yii::app()->user->um->listUsers() as $user)
    echo "email=".$user->email."
";
```

LISTAR USUARIOS QUE TENGAN UN ROL ASIGNADO

(Entrega un `DataProvider`)

```
Yii::app()->user->um->searchUsersByAuthItem('agent',20 /*, array defaultOrder opcional */);
```

<https://bitbucket.org/christiansalazarh/cruge/commits/2d2bb83f9edbbb67d20a0dd7ae2a415018b287bb>

```
<h3>Your Agents:</h3>
<?php
$this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'agents-grid',
    'dataProvider'=>Yii::app()->user->um->searchUsersByAuthItem('agent'),
    'columns'=>array(
        array('name'=>'username'),
        array('name'=>'email'),
```

```
        array('name'=>'userdescription'), // este "magic" method esta descrito en el commit #f0a6268
    ),
));
?>
```

para saber mas sobre "userdescription", leer aqui: <https://bitbucket.org/christiansalazarh/cruge/commits/ceb8b3bcfe3b69dceb07b18648181f558ffd67ec>

LISTAR USUARIOS CON CAMPOS PERSONALIZADOS (ARRAY O DATAPROVIDER)

```
Yii::app()->api->user->um->listUsers
```

Permite emitir tanto un array como un dataprovider, segun los argumentos dados (mira la documentacion del metodo).

Por ejemplo, en una columna de un CGridView podrias listar una columna con el apellido (un campo personalizado) haciendo algo como esto:

```
array('value'=>$data->getCustomFieldValue('\apellido\'))
```

Tambien dispones de:

```
Yii::app()->api->user->um->listAllUsersDataProvider
```

en ambos casos mira la documentacion del metodo para saber mas.

ARMANDO UN COMBOBOX CON LA LISTA DE USUARIOS REGISTRADOS MOSTRANDO LOS CAMPOS PERSONALIZADOS (EJEMPLO)

```
$comboList = array();
foreach(Yii::app()->user->um->listUsers() as $user){
    // evitando al invitado
    if($user->primaryKey == CrugeUtil::config()->guestUserId)
        break;
    // en este caso 'firstname' y 'lastname' son campos personalizados
    $firstName = Yii::app()->user->um->getFieldValue($user,'firstname');
    $lastName = Yii::app()->user->um->getFieldValue($user,'lastname');
    $comboList[$user->primaryKey] = $lastName.", ".$firstName;
}
echo "Users: ".CHTML::dropDownList('userlist','', $comboList)."<hr/>";
```

EDITAR UN CAMPO PERSONALIZADO (EJEMPLO)

```

$usuario = Yii::app()->user->um->loadUserByUsername('jperez');
$campoNombre = Yii::app()->user->um->getFieldValueInstance($usuario, 'nombre');
$campoNombre->value = "Juanito";
$campoNombre->save();
echo "cambiado a: ".$campoNombre->value;

// la siguiente forma tambien es valida, pero solo dara el valor,
// en cambio getFieldValueInstance devuelve el objeto.
echo Yii::app()->user->um->getFieldValue($usuario, 'nombre');
```

GENERAR UN USERNAME AUTOMATICAMENTE BASADO EN EL EMAIL

<https://bitbucket.org/christiansalazarh/cruge/changeset/a224a7f80dea6fe1c55f2892e619dee3e5772855#Lcomponents/CrugeUserManager.phpT641>

```

echo "su usuario será: ".Yii::app()->user->um->generateNewUsername("pepito@gmail.com");
// si ya existía "pepito" en la lista de usuarios (tomado de "pepito"@"gmail.com)
// cruge generará: "pepito.1" y así sucesivamente hasta dar con un "pepito.N" que no exista
```

CREAR NUEVO USUARIO (recomendado)

mas arriba hay otro ejemplo parecido, pero esta funcion es mas fuerte porque permite incluir en un array indexado los valores incluso de campos personalizados. <https://bitbucket.org/christiansalazarh/cruge/changeset/a224a7f80dea6fe1c55f2892e619dee3e5772855#Lcomponents/CrugeUserManager.phpT674>

```

$values = array(
    'username' => 'juanito',
    'email' => 'juanitoperez@gmail.com',
    'nombre' => 'Juanito', // VALOR DE CAMPO PERSONALIZADO
    'apellido' => 'Perez', // VALOR DE CAMPO PERSONALIZADO
    'cedula' => '1298912891', // VALOR DE CAMPO PERSONALIZADO
);
$usuario = Yii::app()->user->um->createNewUser($values);
```

CREAR NUEVO USUARIO (otra forma)

```
public function actionAjaxCrearUsuario(){
    // así se crea un usuario (una nueva instancia en memoria volátil)
    $usuarioNuevo = Yii::app()->user->um->createBlankUser();
    $usuarioNuevo->username = 'username1';
    $usuarioNuevo->email = 'username1@gmail.com';
    // la establece como "Activada"
    Yii::app()->user->um->activateAccount($usuarioNuevo);
    // verifica para no duplicar
    if(Yii::app()->user->um->loadUser($usuarioNuevo->username) != null)
    {
        echo "El usuario {$usuarioNuevo->username} ya ha sido creado.";
        return;
    }
    // ahora a ponerle una clave
    Yii::app()->user->um->changePassword($usuarioNuevo,"123456");

    // IMPORTANTE: guarda usando el API, la cual hace pasar al usuario
    // por el sistema de filtros, por eso user->um->save()
    //
    if(Yii::app()->user->um->save($usuarioNuevo)){
        echo "Usuario creado: id=".$usuarioNuevo->primaryKey;
    }
    else{
        $errores = CHtml::errorSummary($usuarioNuevo);
        echo "no se pudo crear el usuario: ".$errores;
    }
}
```

CREAR USUARIOS FICTICIOS (ALEATORIOS)

Cruge te permite crear usuarios ficticios para tus pruebas, usando nombres y apellidos en inglés. Por defecto se asume que tienes campos personalizados 'firstname' y 'lastname' aunque puedes especificar tus propios campos personalizados para el nombre y el apellido. Considera la precaución de que si este proceso dura mucho entonces PHP puede bloquearse debido a 'max_execution_time', la cual deberás alargar si pretendes crear muchos usuarios a la vez. Lo más recomendable es hacerlo en lotes de a 100.

```
// en esta modalidad crear 100 usuarios con nombres ficticios
// se les asignará el ROL definido en las variables de sistema
// el nombre y apellido ficticio se asignará a campos personalizados
// llamados por defecto: 'firstname' y 'lastname' respectivamente.
Yii::app()->user->um->createRandomUsers(100);

// en esta modalidad se crearán usuarios también pero
// usando los campos personalizados 'nombre' y 'apellido'
//
Yii::app()->user->um->createRandomUsers(100,'default','nombre','apellido');
```

LOGIN USER

<https://bitbucket.org/christiansalazarh/cruge/changeset/fefa510a6eced169a9223ac12c8a093c334c13b1#Lcomponents/CrueUserManager.phpT747>

```

$usuario = Yii::app()->user->um->loginUser('juanito'); // su username o su email..
if($usuario != null){
    // se le ha iniciado una sesión a juanito, tal cual hubiese iniciado sesion manualmente via forms.
}

```

REMOTE LOGIN

<https://bitbucket.org/christiansalazarh/cruge/changeset/cbca2c9820527e005994a9d4f6c9dbea8251115a#Lcomponents/CrueUserManager.phpT789>

- A diferencia de `Yii::app()->user->um->loginUser`, esta función **remoteLoginInterface** puede incluso registrar al usuario e iniciarle la sesión de inmediato (según el argumento "modality" que se le pasa).
- Puede incluso pasarle valores de inicialización mapeados tal cual los hubiese entregado un sistema remoto como facebook o google.
- La función devuelve la URL a la cual deberíamos redirigir al usuario (si el usuario ya existía o se registro automáticamente se le iniciará la sesión de forma automática pudiendo usar tu aplicación web en el acto), esta url puede ser NULL (si hubo fallo) o la url de registration (si modality es "manual") o la url de login (si modality es "auto", la cual asegura que el usuario se ha creado aunque no haya existido antes).

```

$fieldMap = array(
    // username podria estar aqui, pero si no está se autogenerará en base al email
    // aunque podria ponerse asi: 'username'=>'id' (supon que facebook o google envian un campo 'id' )
    'email'=>'contact/email' // se nos envia 'contact/email', pero internamente se llama 'email'
    , 'nombre'=>'first_name' // se nos envia 'first_name', pues lo mapeamos al campo personalizado 'nom
    , 'apellido'=>'last_name' //
);
// los datos que nos mandan con variables propias de facebook o google:
$values = array(
    'email'=>'juanito@gmail.com'
    , 'first_name'=>'Juanito',
    , 'last_name'=>'Perez',
);
$mod = 'auto'; // podria ser auto, manual o none. auto=registra al usuario si no existe.
$resultado = Yii::app()->user->um->remoteLoginInterface($fieldMap,$values,$mod,$info);
if(!$resultado){
    echo "algo ha fallado: ".$info;
}else{
    // si modality es 'auto' la variable $info nos informara con el valor 'registration'
}

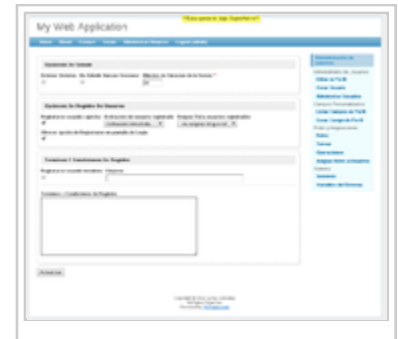
```

```
// en el caso de que haya tocado registrar al usuario porque no existía.
// en tal caso al usuario ya se le ha iniciado la sesion automaticamente
// con Yii::app()->user->um->loginUser
$this->redirect($resultado);
}
```

VARIABLES DEL SISTEMA

Las variables controlan el comportamiento de Cruge, por ejemplo, la pantalla de Registro, detener el sistema, duración de la sesión etc. Las variables del sistema se pueden modificar mediante el menu de administración de usuarios en el sub item 'Variables del Sistema'.

Las variables del sistema estan contenidas en el modelo /cruge/models/data/CrugeSystem.php, también puedes acceder a las variables del sistema mediante API así:



```
if(Yii::app()->user->um->getDefaultSystem()->getn('registerusingcaptcha'))
```

Puedes editar las variables en línea mediante la URL:

```
<?php echo Yii::app()->user->ui->userManagementAdminLink; ?>
```

En este menu verás las siguientes opciones:

1. **Detener Sistema.** si esta activada no se permitira iniciar sesion o usar el sistema en general.
2. **No Admitir Nuevas Sesiones.** solo bloquea el login, es decir nuevas sesiones de usuario.
3. **Minutos de Duración de la Sesión.** el tiempo en minutos en que una sesion expira.
4. **Registrarse usando captcha.** si esta activada presenta un captcha al momento de registrar a un nuevo usuario.
5. **Activación del usuario registrado.** opciones para aplicar al nuevo usuario registrado, puedes activarlo inmediatamente, o mediante activacion manual por

parte de un administrador, o mediante un correo de verificación.

6. Asignar Rol a usuarios registrados. es el rol que tu quieres que se le asigne por defecto al usuario recién registrado.

7. Ofrecer la opción de Registrarse en la pantalla de Login. Esta opción habilita el link de "registrarse" en la pantalla de login. para que esta opción funcione necesitas actualizar tu base de datos con el siguiente script sql:

```
alter table cruge_system add registrationonlogin integer default 1;
```

8. Registrarse usando terminos. si lo activas, el usuario que se va a registrar debe aceptar los terminos que tu indiques.

9. Etiqueta. si la opción anterior esta activa se le hará una pregunta, aquí puedes escribir esa pregunta, por ejemplo: "Por favor lea los terminos y condiciones y aceptelos para proceder con su registro"

10. Términos y Condiciones de Registro. El texto de las condiciones para registrarse, solo aplica si la opción de "registrarse usando terminos" esta activada.

MANEJANDO ENLACES

Puedes invocar al API UI de Cruge para acceder a los enlaces. Esto es importante debido a la complejidad de anidamiento de Cruge debido a que es un módulo, por tanto para ayudarte y evitarte errores es mejor que accedas a las URL de Cruge usando el API.

```
Yii::app()->user->ui (cuidado: user->ui no es lo mismo que user->um, son API's distintas)
```

Lista de Enlaces listos para usar:

```
<?php echo Yii::app()->user->ui->loginLink; ?>
<?php echo Yii::app()->user->ui->logoutLink; ?>
<?php echo Yii::app()->user->ui->passwordRecoveryLink; ?>
<?php echo Yii::app()->user->ui->userManagementAdminLink; ?>
<?php echo Yii::app()->user->ui->registrationLink; ?>
```


debido a características de Yii Framework, estas dos formas son equivalentes:

```
<?php echo Yii::app()->user->ui->getLoginLink('iniciar sesion'); ?>
<?php echo Yii::app()->user->ui->loginLink; ?>
```

para obtener solo la URL y no un objeto generado con CHtml::link (lo cual es lo que devuelven las funciones de Yii::app()->user->ui que terminan en 'Link'), se dispone además de la misma función pero en forma URL, por tanto puedes obtener la URL así:

```
<?php $miUrl = Yii::app()->user->ui->getLoginUrl(); ?>
```

la cual te retornará una Url lista (tal cual como aquellas devueltas por CHtml::normalizeUrl) para que las uses donde necesites.

Presentando el menú de administración de usuarios

Para facilitar el trabajo Cruge trae un menu item con todas las funciones listas en un menú, con solo pasarle este menu a un componente CMenu verás que automaticamente se crea una lista con todas las funciones necesarias para administrar a los usuarios:

```
<?php
// $items sera una array listo para insertar en CMenu, BootNavbar o similares.
$items = Yii::app()->user->ui->adminItems;
?>
```

Usando Bootstrap:

```
<?php
$this->widget('bootstrap.widgets.BootNavbar', array(
    'fixed'=>false,
    'brand'=>"Tu App",
    'items'=>array(
        array(
            'class'=>'bootstrap.widgets.BootMenu',
            'items'=>array(Yii::app()->user->ui->adminItems
        ),
    )); ?>
```



También dispones de esta función del API de Cruge:

```
Yii::app()->user->ui->userManagementAdminUrl
```

La cual brinda toda la lista de actividades a realizarse, de hecho se utiliza como punto de ejemplo tras instalar Cruge:

```
<?php $this->widget('zii.widgets.CMenu',array(
    'items'=>array(
        array('label'=>'Home', 'url'=>array('/site/index')),
        array('label'=>'About', 'url'=>array('/site/page', 'view'=>'about')),
        array('label'=>'Contact', 'url'=>array('/site/contact')),
        array('label'=>'Administrar Usuarios'
            , 'url'=>Yii::app()->user->ui->userManagementAdminUrl // <-- AQUI VA
            , 'visible'=>!Yii::app()->user->isGuest),
        array('label'=>'Login'
            , 'url'=>Yii::app()->user->ui->loginUrl
            , 'visible'=>Yii::app()->user->isGuest),
        array('label'=>'Logout ('.Yii::app()->user->name.')'
            , 'url'=>Yii::app()->user->ui->logoutUrl
            , 'visible'=>!Yii::app()->user->isGuest),
    ),
); ?>
```

USO DE LAYOUTS

Cruge te permite que su interfaz de usuario pueda ajustarse a tu sitio web usando lo que en Yii se conoce como Layouts. Por favor conoce mas acerca del uso de layouts en el sitio oficial de Yii Framework, es importante que lo conozcas.

Básicamente un "Layout" en Yii significa que es un código que va a contener a otro. En este caso, cuando tu usas tu aplicación Yii por defecto se te entrega un layout llamado "main" y otro llamado "column1". Cuando tu usas una vista en Yii su controlador va a poner un layout: `$this->layout = '\\Layouts\\main'`, pero a su vez main incorpora a "column1". Por tanto tu vista va a ser renderizada con "column1" y luego finalmente con "main". Usando este principio simple verás que a Cruge puedes decirle con qué layout quieres que renderize sus vistas, por esta única y básica razón te digo que: No es necesario en lo absoluto modificar las vistas internas de Cruge porque son controlables con layouts.

Hay personas que modifican estas vistas, es un error, porque si mas adelante salen actualizaciones de Cruge no las podrán aprovechar, y en el mejor de los casos podrán causar problemas a Cruge y a toda la aplicación.

Ejemplo:

Supón que quieres que el formulario de registro de nuevo usuario se presente en un esquema de diseño distinto al que yii trae por defecto, entonces tú podrías crear un nuevo layout que se ajuste a tus necesidades y luego indicarle a Cruge mediante la configuración del componente cuál sería ese layout a usar cuando un usuario quiera registrarse, se hace así (ejemplo con bootstrap):

```
<?php
'components'=> array(
    'loginLayout'=>'//layouts/bootstrap',
    'registrationLayout'=>'//layouts/bootstrap',
    'activateAccountLayout'=>'//layouts/bootstrap',
    'generalUserManagementLayout'=>'//layouts/bootstrap',
),
?>
```

Te cuidado especial con "generalUserManagementLayout": este es un layout especial que requiere que dentro de él haya un Portlet, éste último es para presentar las funciones de administracion de usuarios. Para ello por defecto Cruge apunta este valor a: "ui" el cual es el nombre de un layout prefabricado que ya trae un Portlet, practicamente idendico al que Yii trae por defecto llamado //layouts/column2. El Layout para UI de Cruge por defecto es:

```
tuapp/protected/modules/cruge/views/layouts/ui.php
```

Como escribí mas atrás, en este layout (ui.php) hay un Portlet, que será llenado con los items de administración en línea de Cruge, estos items salen del modulo UI de Cruge, el cual es accesible usando:

```
Yii::app()->user->ui->adminItems
```

CRUGE RBAC

El RBAC que Cruge implementa es el mismo de Yii Framework de fábrica, salvo algunos agregados que he notado que han sido incorporados en las nuevas versiones de Yii. RBAC Es un sistema de control de acceso basado en roles (por sus siglas en ingles). Todo el mecanismo RBAC puede ser manejado mediante la interfaz (UI) de Cruge, o mediante su API.



Las dos modalidades para usar en este mecanismo son:

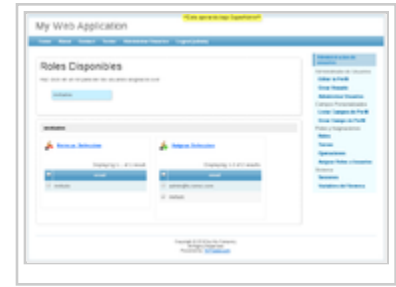
- Usar `checkAccess` para verificar si el usuario tiene permiso.
- Controlando el Acceso a un "controller" o a un "action" usando `CrugeAccessControlFilter`

Usar `checkAccess` para verificar si el usuario

tiene permiso

Es basicamente el mismo mecanismo que provee Yii, pero en Cruge se ha ampliado un poco mas. Para usar este mecanismo: en cualquier parte de tu codigo fuente puedes poner lo siguiente:

```
<?php
if(Yii::app()->user->checkAccess('puede_ver_menu_sistema')) {
...mostar menu sistema...
}
?>
```



(este mecanismo "directo" fue incorporado en las mas recientes versiones de Yii, para la fecha de salida de Cruge al aire no estaba disponible para Yii estándar, antes había que acceder mediante el componente AuthManager de Yii)

Controlando el Acceso a un "controller" o a un "action" usando CrugeAccessControlFilter

Este mecanismo es muy útil, porque permite controlar el acceso a tus controllers/actions de forma totalmente automatizada y controlada mediante la UI de Cruge. Para usar este mecanismo, necesitarás incluir en tu Controller (cualquiera que tú uses y que desees controlar a nivel de permisos) el siguiente código:

```
<?php
..cuerpo de tu controladora...
public function filters()
{
    return array(array('CrugeAccessControlFilter'));
}
..cuerpo de tu controladora...
?>
```

Al usar CrugeAccessControlFilter estas permitiendo que Cruge controle el acceso tanto al "controller" en general como al "action" específico. Ejemplo:

Un usuario cualquiera, incluso un invitado, intenta acceder a la siguiente URL:

```
index.php?r=empleado/vernomina.
```

pues bien, Cruge verificara dos cosas:

- ```
a) el acceso a la controladora: 'Empleado'.
b) el acceso al action 'Vernomina'.
```

lo hara de esta forma:

- ```
a) verifica si el usuario (aun invitado) tiene asignada la operacion: 'controller_empleado'  
b) verifica si el usuario (aun invitado) tiene asignada la operacion: 'action_empleado_vernomina'
```

si ambas condiciones se cumplen (a y b) entonces tendrá acceso al action.

- **Si tu quieres denegar el total acceso a un controller** simplemente no le asignas al usuario la operacion que tenga el nombre del controller antecedido de la palabra 'controller_'.
- **Si tu quieres denegar el acceso a un action de un controller** simplemente no le asignas al usuario la operacion que tenga el nombre del action: 'action_nombrecontroller_nombreaction'.

Saber por donde paso un usuario

Un sistema controlado por Cruge mediante el uso de CrugeAccessControlFilter causara que el usuario (o incluso el invitado) pase por:

```
CrugeWebUser::checkAccess
```

Invocado cuando CrugeAccessControlFilter es el controlador de permiso de tu Controller:

```
CrugeAccessControlFilter::preFilter
```

Programacion del RBAC: diferencias con otras extensiones.

En el caso de Cruge, la programacion de RBAC no se hace "asumiendo que un usuario va pasar por ahi.." (modo Yii Rights). En cambio en Cruge, debes "tratar de hacer pasar al usuario por donde quieres", este ultimo metodo, a mi juicio, es mas seguro porque te obliga a verificar que realmente el usuario pudo acceder o no a tal o cual parte.

Modo de Programación del RBAC

Para activarlo, en la configuración de tu aplicación web (config/main.php) debes poner a "true" estos dos argumentos que explico en detalle:

```
'rbacSetupEnabled'=>true,
```

Permitirá que las operaciones se vayan creando (valor true) en la tabla de la base de datos a medida que vas probando el sistema, de lo contrario deberas crear las operaciones a mano. Las operaciones que se crearan automaticamente seran: 'controller_nombredecontroller' y 'action_tucontroller_tuaction'.

¿ Qué sucede si rbacSetupEnabled esta en valor false ?

Explico con un ejemplo. Si el valor esta en false y creas una nueva controladora o un nuevo action en una existente, entonces el sistema RBAC dirá que -no tiene permiso- para ese action o controladora. Para solucionar esto tienes dos vias: creas la operacion a mano, o pones a true el valor de rbacSetupEnabled y tratas de usar ese controller o action para que sea Cruge quien registre esa operacion, posteriormente tu deberás asignar esa nueva operación al rol de los usuarios.

Lee acerca de CrugeAccessControlFilter para que sepas como pedirle a Cruge que controle tu controladora y action. [[1] (http://yiiframeworkenespanol.org/cruge/index.php?title=P%C3%A1gina_Principal#Controlando_el_Acceso_a_un_controller_o_a_un_]

```
'allowUserAlways'=>true,
```

Permitirá el paso (valor true) al usuario aunque este no tenga permiso. Cuando estás en producción ponlo en 'false' lo que causara que el usuario reciba una excepción si no dispone del permiso para usar cualquier rol, tarea u operación. Es bueno ponerlo en TRUE para el momento en que estas configurando los permisos.

Desplegando una lista de permisos requeridos al pie de tu página

Aunque allowUserAlways tenga valor TRUE (el cual hara que cruge siempre de paso con checkAccess) siempre podrás conocer que permisos "fallaron" (permisos que se requieren para que el usuario pueda usar una parte del sistema), puedes agregar a tu layout principal una llamada a displayErrorConsole, asi:

```
<?php echo Yii::app()->user->ui->displayErrorConsole(); ?>
```

¿ y esto para qué sirve ?

Sirve cuando estas diseñando los permisos de tus usuarios. Pones `allowUserAlways` en `true`, luego vas haciendo pasar al usuario con el rol a programar por todo el sistema requerido e irás viendo al pie de la página la lista de permisos que se han requerido. En un browser paralelo abres el administrador de cruge con tu superusuario y vas aprobando los permisos que se requieran.

Es una herramienta de programación que no estará disponible cuando `rbacSetupEnabled` sea `false`, pero no tienes de que preocuparte, cuando este sea el caso simplemente no imprimira nada, por tanto puedes dejarla en tu layout con seguridad de que no va a molestar.

Usando el LOG

Adicionalmente todos los errores de permiso que se generen seran reportados en el log bajo el key 'rbac', para poder visualizar los errores en `protected/runtime/application.log` deberas configurar tu `config/main.php` para indicar el key del log:

```
<?php
'log'=>array(
    'class'=>'CLogRouter',
    'routes'=>array(
        array(
            'class'=>'CFileLogRoute',
            'levels'=>'error, info, rbac', // <--- agregar 'rbac'
        ),
    ),
),
?>
```

eso causara que en el archivo de log (`protected/runtime/application.log`) se generen entradas como las que describo a continuación:

```
2012/07/27 15:24:25 [rbac] [application] PERMISO REQUERIDO:
!invitado
iduser=2
!tipo:operacion
!itemName:action_catalog_imageh'
```

Tips de Programacion del RBAC

- Cuando quieras programar el RBAC con cruge, usa dos navegadores: uno

abierto con un usuario administrador, para que puedas ir activando las operaciones para un rol específico a medida que sea necesario, y abre otro navegador con el usuario que tenga el rol que quieres programar. No olvides tener activado el flag: `rbacSetupEnabled`.

- Por ejemplo, quieres que el usuario 'juan' que tiene asignado el rol 'empleado_regular' tenga solo acceso solo a donde quieres, entonces en el segundo navegador inicia sesión con 'juan', y tratas de ir al menú u operación requerida, cruge irá informando al pie de la página los permisos requeridos. Luego con el navegador que tiene abierto el usuario 'admin' entonces vas verificando los permisos y se los asignas al rol.
- Considera que cuando `rbacSetupEnabled` está habilitado, entonces, asumiendo además que no hay ninguna operación creada irás viendo que cruge creará las operaciones automáticamente de acuerdo a donde el usuario 'juan' vaya pasando, solo las crea si previamente no existen.
- Ejemplo: no hay ninguna operación creada, entonces el usuario juan quiere entrar a 'site/index', por tanto, si la controladora 'siteController' está manejada con el filtro: 'CrugaAccessControlFilter' (ver tema más arriba) entonces verás que se creará una operación llamada 'site_controller' y otra 'action_site_index', deberás entonces asignarle estas dos operaciones al rol al cual 'juan' pertenece.

Superusuario e Invitados

Seleccionando el modo de inicio de sesión: Por usuario, por Email, o ambos

Cruga permite que puedas seleccionar la modalidad de inicio de sesión, se configura en `config/main.php`.

```
'availableAuthModes'=>array('username','email'),
```

Esto causará que cruge etiquete el formulario de login con "Username o Email:", pudiendo introducirse tanto un username como un email. Internamente el filtro de autenticación de Cruga probará primero con username, y luego con email.


```
'availableAuthModes'=>array('username'),
```

Esto causará que cruge etiquete el formulario de login con "Username:", Solo aceptara un login válido con su username. Internamente el filtro de autenticacion de Cruge probará solo con username.

```
'availableAuthModes'=>array('email'),
```

Esto causará que cruge etiquete el formulario de login con "Email:", Solo aceptara un login válido con su email. Internamente el filtro de autenticacion de Cruge probará solo con username. **IMPORTANTE:** si habilitas esta opción e inicias sesión como "admin" no funcionará, deberás poner: "admin@tucorreo.com", el cual obviamente podrás cambiar luego.

El superUsuario

Por defecto, cruge considera a un superusuario, para proposito de administracion, depuracion etc. Para que cruge considere a un usuario como un "superusuario" entonces éste deberá tener un "username" con el mismo valor configurado en CrugeModule::superuserName.

Por defecto este valor viene configurado como 'admin' (con password admin).

Para cambiar este valor, al igual que otros parametros de Cruge, no debes cambiar nada en CrugeModule, sino mediante edición de "tuapp/protected/config/main.php":

```
'cruge'=>array(
    'superuserName'=>'administrador', (suponiendo que no te gusta 'admin')
),
```

El superUsuario para Cruge tiene un caso especial en el método `Yii::app()->user->checkAccess (cruge\components\CrugeWebUser.php)`: **si el usuario activo es un superusuario entonces checkAccess devolverá true -siempre- sin importar ningún rol, tarea u operación.**

Como saber si estamos ante un superusuario ?

Consultando a:

```
Yii::app()->user->getIsSuperAdmin()
```

o mas corto:

```
Yii::app()->user->isSuperAdmin
```

El usuario Invitado

Cruge hace especial tratamiento al usuario invitado. Para esto CrugeModule contiene un atributo llamado 'guestUserId' el cual es usado para indicarle al sistema Cruge cual de sus usuarios existentes en la base de datos de usuarios es el invitado, dicho de otro modo es para reconocer quien es el usuario marcado como invitado. Solo debe haber un solo invitado en una aplicación web, no tiene sentido alguno tener mas de uno.

Por defecto cuando Cruge es instalado, su script de base de datos (protected/modules/cruge/data) creará dos usuarios:

```
insert into `cruge_user`(username, email, password, state) values
  ('admin', 'admin@tucorreo.com','admin',1),('invitado', 'invitado','nopassword',1);
```

Siendo 'admin' el usuario con ID 1, y siendo 'invitado' el usuario con ID 2. Por esto veras que por defecto en CrugeModule.php ya esta predefinido el atributo guestUserId en 2. No lo cambies a menos que cambies el ID del usuario invitado.

Cómo maneja Cruge al usuario invitado.

Por defecto en YII cuando tu llamas a `Yii::app()->user->id` esta devuelve 0 (cero) cuando un usuario es invitado. En Cruge esta misma llamada a `Yii::app()->user->id` devolverá al valor configurado en `CrugeModule::guestUserId`, el cual de fábrica es dos (2). Siempre puedes confiar en `Yii::app()->user->isGuest`, ya que ésta función considera todo esto para saber si el usuario es un invitado.

```
Usa: Yii::app()->user->isGuest para saber si estamos operando ante un invitado.
```

TRAS INSTALAR, DEBES CONFIGURAR EL USUARIO INVITADO

Esto es importante: Si no asignas al usuario invitado a ningún rol (por defecto es así) entonces no tendrá acceso a ningún "controller" que esté siendo manejado por `CrugeAccessControlFilter` (mas arriba explico acerca de `CrugeAccessControlFilter`). Por tanto tras instalar tu sistema con Cruge debes hacer lo siguiente:

- Crea un rol llamado 'invitados'.
- Asigne a ese rol las operaciones necesarias, por ejemplo 'controller_site', 'action_site_index', 'action_site_contact', 'action_site_login' y otras que vayas viendo que se requieran (usa el LOG).
- Asigna este rol creado al usuario invitado usando la página de administración de permisos de Cruge, ó, mediante las "variables del sistema": "Asignar este rol al usuario registrado".

Tip: No necesariamente el rol del 'invitado' debe llamarse 'invitado'. Por conveniencia es sano que así.

Importantísimo a menos que quieras ver siempre un "401 Acceso no Autorizado":

NO USES EL CAMPO "REGLAS DE NEGOCIO", DEJALO EN BLANCO, A MENOS QUE CONOZCAS BIEN COMO ESTAS FUNCIONAN. Un mal uso de este campo causara que `checkAccess` siempre retorne `FALSE` causando un "401" "Acceso No Autorizado" SIEMPRE, sin importar que el rol tenga permiso sobre un action.

Creando el Rol de los Invitados:

*** You are working as Super Administrator


My Web Application

[Home](#) [About](#) [Contact](#) [Administrar Usuarios](#) [Logout \(admin\)](#)

Roles

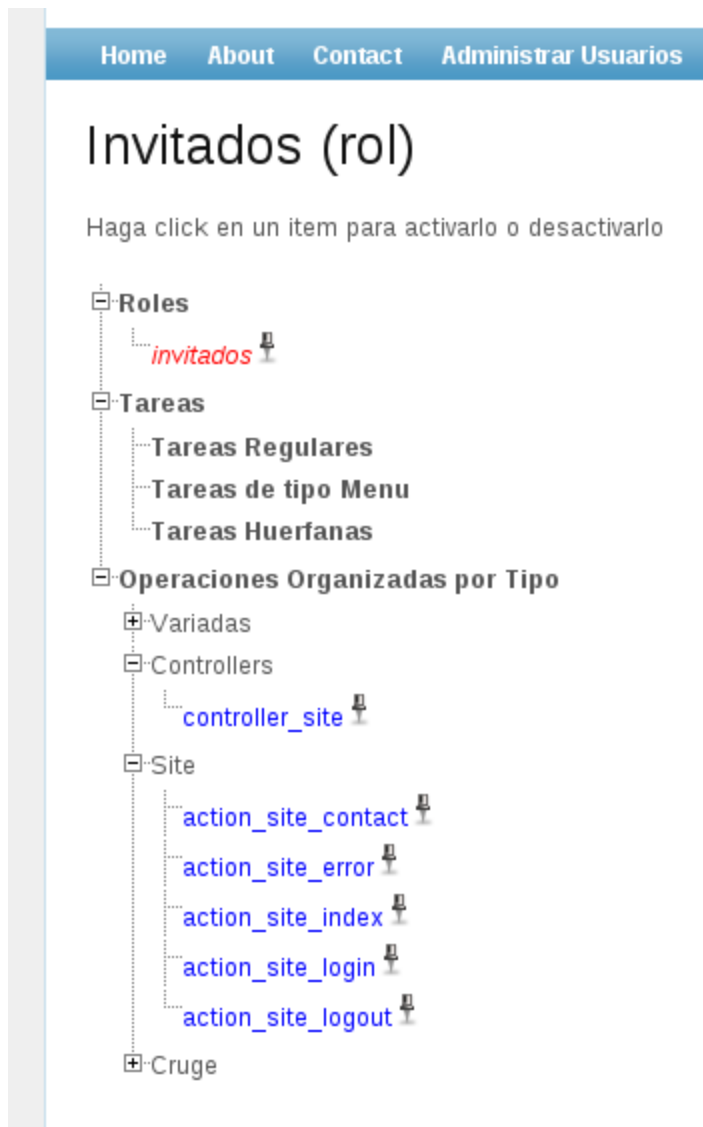
[Crear Nuevo Rol](#)

Displaying 1-1 c
Sort

| |
|---|
| invitados |
| propiedades editar permisos  |

Copyright © 2013 by My Company.
All Rights Reserved.
Powered by [Yii Framework](#).

Dando permisos para que los invitados puedan ver el sitio web. (Solo si en siteController has puesto el filtro CrugeAccessControlFilter).



Manejo de Eventos con Cruge

EVENTOS DE INICIO, CIERRE Y EXPIRACIÓN DE SESIÓN

Puedes hacer que Cruge vaya a una URL específica cuando ocurra uno de estos eventos:

- inicio de sesión
- cierre de sesión
- la sesión expira.

Esto se configura en `tuproyecto/protected/config/main.php`, allí puedes colocar las

URL en las siguientes variables del módulo Cruge:

```
<?php
'afterLoginUrl'=>array('/site/welcome'),
'afterLogoutUrl'=>array('/site/page','view'=>'about'),
'afterSessionExpiredUrl'=>null,
?>
```

(IMPORTANTE: no olvidar el slash inicial "/" sino la url no funcionará)

LOGIN, LOGOUT, SESIONES. CÓMO FUNCIONA EN CRUGE.

Debes comprender que hay dos estados en un sistema RBAC estándar como lo es Cruge:

■ La autenticación

Es el estado del sistema RBAC en el cual se verifica que un usuario es realmente quien dice ser, se verifica mediante un usuario y una clave, un código, un mecanismo de OPEN-ID (pronto será incorporado en Cruge, al igual que Facebook)

■ La sesión.

Tras autenticarse, a un usuario se le asigna un espacio y un tiempo para operar en el sistema. El tiempo de la sesión se configura en las “Variables del Sistema” (ver tema al inicio). Tras cumplirse el tiempo, Cruge abortará al usuario emitiendo el evento `onSessionExpired` (de la clase: `cruge.models.filters.DefaultSessionFilter`), el cual derivará en una llamada a la URL `afterSessionExpiredUrl` (ver arriba). Cruge maneja la sesión bajo una arquitectura delicada que será explicada a futuro con mayor detenimiento y con soporte de diagramas UML, por ahora no viene al caso detallar tanto.

Filtros:

Cruge tiene un 'filtro para otorgar sesiones' y un 'filtro de autenticación' estos funcionan así:

- Primero se pasa por el filtro de autenticación, el cual le da sentido a `'Yii::app()->user->getUser()'`, luego el filtro de sesión verifica si el sistema está apto para recibir una nueva sesión, quizás está en mantenimiento o quizás no, por tanto es el filtro de sesión quien ahora entra en juego.
- Una vez que el filtro de autenticación determina que se puede dar una sesión

a 'juan perez', entonces se le crea una sesion y se llama a un evento llamado 'onLogin' de la clase 'cruge.models.filters.DefaultSessionFilter'.

Este evento de onLogin es quien establece el valor a `Yii::app()->user->returnUrl` el cual es procesado por `UiController` para redirigir el browser a una pagina que tu indicas.

Ahora, importante, cuando tu haces `logOff` manualmente, o si por alguna razón tú usando el api estándar de Yii haces una llamada a `Yii::app()->user->logout()` verás que también serás redirigido a la URL que hayas especificado en `'afterLogoutUrl'`.

Conviviendo con **CAccessControl** filter.

Supongamos que tras iniciar sesion exitosamente quieres que el usuario sea redirigido al `actionBienvenido` de `siteController` (`index.php?r=site/bienvenido`), *pero utilizando el filtro `accessControl` que Yii trae por defecto y no mediante los eventos de sesión que te he mostrado en la página anterior.*

Pues bien el metodo que aquí describiré es algo estándar para Cruge o para Yii en general, no es nada nuevo.

- 1) en `siteController` (en el controller de tu gusto) creas un action el cual desplegara la página que solo verá aquel usuario que haya iniciado sesion exitosamente.

```
<?php
public function actionBienvenido(){
    $this->render('bienvenido');
}
?>
```

- 2) en `siteController` usas el filtro `accessControl` y los rules (que vienen de caja en Yii), asi:

```
<?php
public function filters()
{
    return array('accessControl',);
}
public function accessRules()
{
    return array(
        array('allow',
            'actions'=>array('index','contact','captcha'),
            'users'=>array('*'),),
        array('allow',
            'actions'=>array('bienvenido'),
```

```

'users'=>array('@'),
),
array('deny', // deny all users
'users'=>array('*'),
),
);
}
?>

```

Con esta configuración de CAccessControl le estas diciendo a tu aplicacion que para el action "site/bienvenido" se requiere que el usuario deba haber iniciado sesion exitosamente (con cruge o con yii tradicional). De este modo, si un usuario invitado presiona el enlace "login" entonces tu lo envias a site/bienvenido, si no ha iniciado sesion se le pediran credenciales y luego se le enviara a la vista site/bienvenido. por tanto el siguiente paso es requerido:

- 3) finalmente sustituye tu enlace a login por un enlace a site/bienvenido.

Que sucederá ?

```

* Tu invitado visita tu website (no ha iniciado sesion aun por eso es un invitado) y sigue el enlace 'lo
* Tu invitado sera redirigido automaticamente a "index.php?r=cruge/ui/login" (o a la url de login del s
* Luego tras iniciar sesion exitosamente sera redirigido automaticamente a "index.php?r=site/bienvenido

```

Como has visto Cruge es un sistema orquestado para trabajar en conjunto con el actual sistema estandar de autenticacion de Yii, por eso como dije antes es una extension: porque extiende la funcionalidad basica de Yii a un nivel mas alto.

Ciertos Actions Requieren Inicio de Sesion

Se puede hacer en combinacion con CAccessControl y Cruge.

Para que CAccessControl ? para agregar reglas especiales que Cruge no maneja. **Y Para que Cruge ?** para que controle en base a ROLES el acceso al action. Puede que CAccessControl permita tambien controlar en base a Roles, pero si estas usando Cruge haz que sea Cruge quien lo maneje.

En este extracto de codigo mostro como lograr una combinacion entre CAccessControl y Cruge para hacer que si una persona no ha iniciado sesion vaya a una pagina de login especifica y no la que por defecto el sistema tiene (/cruge/ui/login o /site/login).


```

class AgentController extends PwhBaseController
{
// 1. accessControl, y luego Cruge

    public function filters(){
        return array('accessControl',array('CrugeAccessControlFilter'));
    }

// 2. las reglas para 'accessControl'. Fijaos que le doy un action a "Yii::app()->user->loginUrl".
// Es un truco, para aprovechar que cuando CAccessControl detecte que tal action requiera login
// entonces va a redirigir a ese action indicado alli (user->loginurl), lo que hacemos es forzarlo
// para que vaya a ese action.

    public function accessRules()
    {

        Yii::app()->user->loginUrl = array("/pwh/agent/login");

        return array(
            array('allow','actions'=>array(
                'login','passwordrecovery','captcha'),'users'=>array('*'),),
            array('allow','actions'=>array('profile'),'users'=>array('@'),),
            array('deny','users'=>array('*'),),
        );
    }

    ...
    BLA
    ...
}

```

FILTROS

Cruge permite que se pueda extender más allá usando filtros. Existen varios tipos de filtros, todos se instalan en config/main y disponen de una interfaz (interface) que debes respetar.

La idea principal a cubrir con los filtros es que mediante config/main tu puedas decirle a Cruge cuales filtros va a usar, para que estos hagan el trabajo que tu necesitas pero sin tocar ni modificar el core de Cruge, en cambio usando el concepto de Override de la programación orientada a objetos (POO).

Filtros de sesión: (clic [aquí](#) para instrucciones de implementacion) Permite que puedas controlar como se entrega una sesion a un usuario, inclusive puedes denegarla. Se ubica en:

```
protected\modules\cruge\models\filters\DefaultSessionFilter.php
```

Filtro de usuario: (clic [aquí](#) para instrucciones de implementacion) Permite saber si un usuario actualizo su perfil. Se ubica en:

```
protected\modules\cruge\models\filters\DefaultUserFilter.php
```

Filtros de autenticación: Permite que amplies como se busca un usuario para ser autenticado. No recomiendo alterarlo. Se ubica en:

```
protected\modules\cruge\models\auth\CrugaAuthDefault.php
```

CÓMO SE OTORGA UNA SESION EN CRUGE y COMO ESTA EXPIRA

Cruge maneja las sesiones con un "filtro de sesion", hay dos casos aqui: o usas filtro de sesion por defecto (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/models/filters/DefaultSessionFilter.php?at=master%7Cel>) que cruge provee, el cual ya viene indicado en config/main o implementas este filtro de sesion personalizado.

Cuando tu quieres iniciar sesion cruge busca el filtro de sesion que tiene instalado en config/main, el cual puede ser cualquiera de los dos que te indico en el parrafo anterior. Desde ese filtro se valida que haya o no un registro de sesion activo y lo reutiliza, sino lo crea nuevo.

ASI SE CREA LA SESION:

1. Por defecto en el filtro de sesion por defecto (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/models/filters/DefaultSessionFilter.php?at=master%7Cel>) si de determina que hay que crear una sesion lo hace mediante una llamada a `Yii::app()->user->um->createSession($user, $sys);` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/models/filters/DefaultSessionFilter.php?at=master#cl-71%7C>)
2. cuando se invoca a `createSession` esta extrae el parametro de caducidad de la sesion mediante una llamada a `$sys->getn('sessionmaxdurationmins')` el cual LEE el valor que tu has configurado en la pantalla de variables de sistema desde las funciones del administrador.
3. A traves de `CrugaFactory` se invoca a `CrugaSession::create` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/models/data/CrugaSession.php?at=master#cl-26>) que es en donde finalmente se crea el objeto de sesion con una fecha de expiracion.
4. La fecha de expiracion usada en `CrugaSession::create` se calcula usando una llamada a `makeExpirationDateTime` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/components/CrugaUtil.php?at=master#cl-77>) ubicada en

CrugeUtil la cual dice así:

```
$model->expire = CrugeUtil::makeExpirationDateTime($durationMins);
```

CONTROL AVANZADO DE SESIONES Y EVENTOS DE AUTENTICACION Y SESION

El uso del filtro de sesión conocido como "MiSesionCruge" es el mecanismo oficial de control de sesión cuando usas Cruge. El atributo 'afterLoginUrl', 'afterLogoutUrl' y 'afterSessionExpiredUrl' de la configuración de Cruge solo son usados cuando no hay un filtro de sesión que controle a estos eventos, en otras palabras, cuando hay un filtro de sesión entonces se usará a cambio de lo que hayas puesto en esos atributos de configuración.

El propósito del filtro de sesión MiSesionCruge es controlar el otorgamiento de sesiones y los eventos de inicio, cierre y expiración de una sesión usando la arquitectura OOP de Cruge.

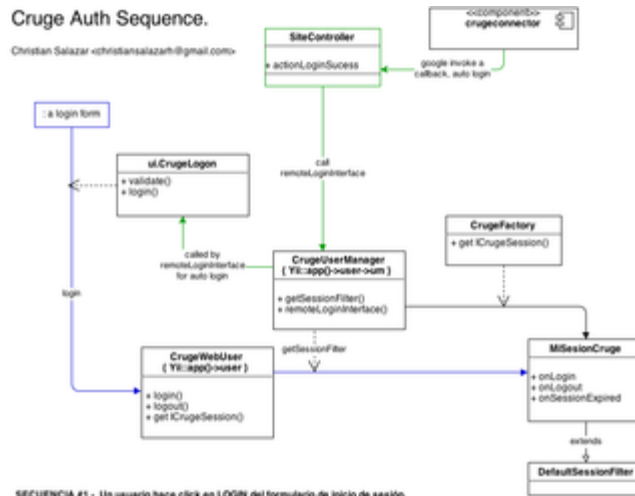
Para esto se hacen dos cosas:

- Se crea una clase personalizada en tu ruta de componentes /tuapp/protected/components/
- Se edita tu archivo de configuración, para indicarle a Cruge que use esta nueva clase en vez de la que usa por defecto.

Cómo Cruge llama a tu clase MiSesionCruge ?

Mediante cualquier invocación a `Yii::app()->user->login()`, `Yii::app()->user->logout()` y cuando la sesión expira tras cualquier consulta a `Yii::app()->user`. Estas invocaciones no las haces tu directamente, sino mediante cualquier formulario de login, uno que hagas tu o el que ya trae cruge, es decir si tu haces tu propio formulario de Login éste invocará finalmente a `Yii::app()->user->login()` y esto invocará a `MiSesionCruge`.

Aquí puedes ver este diagrama que muestra cómo Cruge llama a tu clase



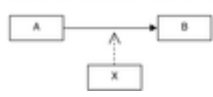
SECUENCIA #1 - Un usuario hace click en LOGIN del formulario de inicio de sesión.

- loginform hace instancia \$model basada en CrugeLogon, para validar credenciales hace model.validate, y tras éxito invoca a model.login()
- model.login() invoca a Yii::app()->user->login(), solicita a CrugeUserManager (getSessionFilter) al filtro de sesión activo.
- CrugeUserManager, le pide al Factory que le de el filtro de sesión registrado en CONFIG MAIN, por tanto esta (Factory) le da una instancia de MiSesionCruge.
- de regreso a CrugeWebUser.login, se usa el filtro para informar el login exitoso.

SECUENCIA #2 - Login automatico tras una exitosa autentificación remota hecha mediante CrugeConnector usando a un proveedor de autentificación (google, facebook, por ejemplo).

- Google ha enviado una llamada de regreso a la aplicación mediante un callback (google-callback.php, ver CrugeConnector doc)
- CrugeConnector invoca a SiteController:actionLoginSuccess
- actionLoginSuccess invoca a CrugeUserManager:remoteLoginInterface (Yii::app()->user->remoteLoginInterface) quien mediante otro metodo privado procesa el login "como si hubiese sido hecho mediante el formulario normal", es decir invocando a CrugeLogon:login()
- CrugeLogon:login() repite el mismo proceso de la SECUENCIA #1 para llegar a MiSesionCruge.

TIP PARA COMPRENDER EL DIAGRAMA:
 Esto se llama "clase de asociación". Indica que: "A se conecta a B mediante el uso de X"



MiSesionCruge:

Manos a la obra:

Edita tu archivo: /tuapp/protected/config/main.php y en la sección "cruge" (dentro de modules) agregas la siguiente linea:

```
'defaultSessionFilter'=>'application.components.MiSesionCruge',
```

Crea el archivo MiSesionCruge.php en: /tuapp/protected/components /MiSesionCruge.php, y copie el código que coloco aqui abajo.

```

<?php
/**
 * Esta clase sirve para personalizar las acciones de inicio y cierre de sesión.
 * requiere que la registres en config/main dentro de cruge setup:
 *
 * 'defaultSessionFilter'=>'application.components.MiSesionCruge',
 *
 * @author Christian Salazar (christiansalazarh@gmail.com)
 */
class MiSesionCruge extends DefaultSessionFilter {

```

```
/**
    este evento es invocado cuando un usuario ha hecho LOGIN EXITOSO.
    Puedes tomar tus propias acciones aqui, no se esperan valores de
    retorno.

    si quieres controlar a un usuario que ha sido autenticado entonces
    deberás trabajar en el método aqui provisto: startSession. mas abajo.
    Una cosa es la sesion otra la autenticacion, aqui solo se notifica que
    un usuario existe.
*/
public function onLogin(/*ICrugeSession*/ $model){
    parent::onLogin($model);
    Yii::log("PASANDO POR ONLOGIN","info");
}

/**
    permite que puedas indicar que hacer antes de cerrar la sesion.
    util porque hay acciones que no podras realizar en onLogout() porque la sesion ya estara
    por tanto aqui.

    disponible desde el commit:
    https://bitbucket.org/christiansalazarh/cruge/commits/60af8068f446639571457fecb6694bb7cd
*/
public function onBeforeLogout(/*ICrugeSession*/ $model) {
    // por ejemplo, podrias determinar el tipo de usuario (segun su rol) y redirigir al usua
    // a esa pagina en onLogin() tras el cierre de sesion (NO AQUI !!)
    //
    return true; // o false para que la sesion no se cierre y vuelvas al action de donde vine
}

/**
    este evento es invocado cuando un usuario ha hecho logout, o cuando
    explicitamente se invoca a
        Yii::app()->user->logout.
    Puedes tomar tus propias acciones aqui, no se esperan valores de
    retorno.
*/
public function onLogout(/*ICrugeSession*/ $model) {
    parent::onLogout($model);
    Yii::log("PASANDO POR ONLOGOUT","info");
}

/**
    este evento es invocado cuando una sesion ha expirado. no se esperan
    valores de retorno, solo puedes colocar aqui tus propias acciones.
*/
public function onSessionExpired(/*ICrugeSession*/ $model) {
    parent::onSessionExpired($model);
    Yii::log("PASANDO POR ONSSESSIONEXPIRED","info");
}

/**
    Este metodo es invocado por el core de Cruge cuando se requiere una
    nueva sesion para un usuario que ha iniciado sesión. El proposito aqui
    es que tu puedas tomar tus propias acciones y decisiones al momento de
    otorgar una sesion a un usuario, pudiendo revocarla si lo deseas
    usando a:
        CrugeSession::expiresession()

    Por defecto es altamente recomendado que retornes:

        return parent::startSession($user, $sys);

```

Lo que aquí se espera es que se retorne una nueva instancia de un objeto que implemente la interfaz ICrugeSession (por defecto la clase CrugeSession lo hace y normalmente es la instancia que aquí se retorna)

la implementación base de startSession usará las siguientes funciones del API para hallar y crear una sesión según sea el caso:

```
$sesion = Yii::app()->user->um->findSession($user);
y
$sesion = Yii::app()->user->um->createSession($user,$sys);
```

para caducarla de inmediato usas:

```
$sesion->expiresession();
```

y luego invoca a :

```
$this->onSessionExpired();
```

para otorgar una sesión al usuario se hacen por defecto validaciones contra el estado sistema, la caducidad de la sesión y otras cosas de relevancia.

```
*/
public function startSession(/*ICrugeStoredUser*/ $user,/*ICrugeSystem*/ $sys) {
    Yii::log("PASANDO POR startSession","info");
    return parent::startSession($user, $sys);
}
}
```

¿ Cómo Redirigir al usuario a una página específica tras iniciar sesión ?

Primero hay que incorporar la nueva clase MiSesionCruge (ver arriba) y en el método onLogin insertar lo siguiente:

```
public function onLogin(/*ICrugeSession*/ $model){
    parent::onLogin($model);
    if(Yii::app()->user->isSuperAdmin)
        Yii::app()->getController()->redirect(array("/admin/index"));
}
```

El código antes descrito redirigirá al usuario al acción "admin/index" si detecta que es un superadmin, pero es solo un ejemplo, aquí puedes insertar cualquier código, cualquier condición y redirigir al usuario a cualquier parte.

CONTROL AVANZADO DE EVENTOS DE INSERCIÓN Y ACTUALIZACIÓN DE USUARIOS

Cruge provee un filtro de usuario (<https://bitbucket.org/christiansalazarh/cruge>)

/src/d9f9f60a01ea/models/filters/DefaultUserFilter.php?at=master) que te permite tomar acciones ante los cambios que puedan requerirse en los datos de un usuario, a diferencia del filtro de sesión el cual te permite tomar alguna acción cuando la sesión inicia o se cierra.

Cruge por defecto usa `cruge/models/filters/DefaultUserFilter.php` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/models/filters/DefaultUserFilter.php?at=master>) debido a que está configurado de fábrica en los atributos de `[CrugeModule.defaultUserFilter` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01eaf9e2fec3810284d043df06e5afa5/CrugeModule.php?at=master#cl-83>)] debido a que este argumento tiene este valor:

```
/*CrugeModule.php */ public $userFilter='cruge.models.filters.DefaultUserFilter';
```

¿ Cómo implementar tu propio filtro de usuario ?

PASO 1) crea una clase:

```
/protected/components/MiFiltroUsuario.php
```

PASO 2) en esta clase escribe lo siguiente:

```
<?php
class MiFiltroUsuario implements ICrugeUserFilter
{
    public function canInsert(ICrugeStoredUser $model)
    {
        // si hay algun error, retornar false y reportar el error asi:
        // $model->addError('', 'algun error');
        return true;
    }

    public function canUpdate(ICrugeStoredUser $model)
    {
        // si hay algun error, retornar false y reportar el error asi:
        // $model->addError('', 'algun error');
        return true;
    }
}
?>
```

PASO 3) indícale a Cruge que quieres que use esta nueva clase en vez de la de fábrica, se hace editando a: `protected/config/main.php`

```
.....  
'modules'=>array(  
  'cruge'=>array(  
    ...  
    'userFilter' => 'application.components.MiFiltroUsuario',  
    ...  
  )  
),  
.....
```

LISTO. Verás que ahora cuando insertes un nuevo usuario o cuando un usuario modifique su perfil, lo sabrás (e incluso podrás denegarlo) utilizando los metodos que la interfaz indica en tu clase MiFiltroUsuario.

¿ Cómo se usan esos métodos ?

El método canInsert(ICrugeStoredUser \$model) recibe una instancia del usuario que se quiere crear, cuidado, ese usuario NO EXISTE solo se te esta preguntando si quieres aceptarlo.

El método canUpdate(ICrugeStoredUser \$model) recibe una instancia del usuario que se quiere actualizar, se te esta preguntando si quieres aceptar los cambios.

En ambos casos tu podrias provocar errores de validación a proposito usando a `$model->addError('algun error');` para forzar a que el usuario cumpla algunas reglas extra de validación que quisieras imponerle.

¿ Quien y cuando se invoca a este filtro ?

Se invoca cuando se hace una llamada a `Yii::app()->user->um->save($model)` (<https://bitbucket.org/christiansalazarh/cruge/src/d9f9f60a01ea/components/CrugeUserManager.php?at=master#cl-332>) . Cuando un usuario va a ser insertado o actualizado siempre se invoca a este metodo, por eso siempre insisto en que cuando se requiera actualizar a un usuarios se haga mediante la llamada a `Yii::app()->user->um->save`, para que asi se respete al sistema en general.

EL ENVIO DE CORREOS CON CRUGEMAILER

(puedes ver el tema ampliado haciendo click en: CrugeMailer)

CrugeMailer no solo esta hecho para Cruge, sino para proveer una plataforma de email a TODA TU APLICACION.

- Permite que tu aplicación envíe correos basados en vistas.
- Centraliza el método de envío de email.
- Convierte al correo en un componente de negocio.

Ejemplo:

```
Yii::app()->crugemailer->enviarEmailNuevaClave($usuario);
```

(es un ejemplo, la función 'enviarEmailNuevaClave' no existe en Cruge, pero si verás otras similares)

Primero es necesario que comprendas que no existe un método público de crugemailer al cual llamar para enviar un correo, me refiero a que -no verás- llamadas como esta:

```
Yii::app()->crugemailer->sendEmail("hola","alguien@gmail.com","body"); // asi no se usa CrugeMailer
```

En cambio si verás llamadas muy específicas que tu mismo creas en una clase derivada de CrugeMailer, por ejemplo:

```
Yii::app()->crugemailer->dimeHola("hola",$alguien); // esta si es la manera de usar CrugeMailer
```

Lo que se hace es que internamente dentro de esa función de ejemplo 'dimeHola' se invocará a una vista que recibirá como argumento el texto "hola" (usando las funciones que CrugeMailer provee), se imprimirá el contenido del mensaje sobre una vista para posteriormente enviársela por correo a \$alguien.

Podrías preguntarte ¿ Y cómo selecciono que método será usado para enviar el correo ? mail() ? phpmailer() ? otro() ?

Es decisión tuya: en esa clase derivada de CrugeMailer que tu debes crear, sobrescribirás el método sendEmail para especificar (o implementar) el método de tu preferencia.

Siguiendo el ejemplo de arriba, es dentro del método 'dimeHola' de tu clase extendida de CrugeMailer donde se especifica cómo se enviará ese correo, si por mail() o por otro método de tu preferencia.

Comprende que a la aplicación entera no le compete este detalle de cómo se envía el email, a la aplicación le interesa "decirle hola a alguien". Este es concepto tras CrugeMailer, y es lo que Cruge usa. Esto es el concepto de Encapsulamiento de la OOP hecho práctica.

Configurar CrugeMailer

```
'crugemailer'=>array(
'class' => 'application.modules.cruge.components.CrugeMailer',
'mailfrom' => 'email-desde-donde-quieres-enviar-los-mensajes@xxx.com',
'subjectprefix' => 'Tu Encabezado del asunto - ',
'cc' => 'unacopiaparami@xxx.com',
'bcc' => 'unacopia_oculta_parami@xxx.com',
'replyTo' => 'que_me_respondan_a_mi_en_vez_de_a_mailfrom@xxx.com',
),
```

Log: para ver el log de los correos que se envian, debes agregar el keyword "email" a la configuracion de CLogger en tu config/main, con eso podras ver los mensajes marcados como "email" que contendrán siempre lo que se ha enviado.

Leer: <https://bitbucket.org/christiansalazarh/cruge/commits/15ebd9dd7c27c6aa407fbe29db9813711308940b>

El argumento class es quien cuál es la clase que implementará al componente crugemailer, por tanto si creas una nueva clase derivada de CrugeMailer (que extiende de..), deberás poner ahí la nueva ruta y toda la aplicación web incluso cruge se aprovecharán de este cambio.

Si en esta clase derivada sobrescribes el metodo sendMail verás que toda la aplicación enviará correos mediante ese metodo que tu seleccionas.

Personalizar CrugeMailer

Supongo que quieres crear un nuevo tipo de correo basado en una vista para tu aplicación, digamos que quieres algo como:

```
Yii::app()->crugemailer->notificarStockMercancia($usuarioSeleccionado,"stock va a vencer producto xxx");
```

1. Creas una clase que extienda de "CrugeMailerBase", la llamarías por ejemplo: "MiClaseCrugeMailer" y supongamos que la guardas en tu directorio de 'protected/components/MiClaseCrugeMailer.php' (!!no dentro de cruge module!!)

```
<?php
// esta clase va en: 'protected/components/MiClaseCrugeMailer.php' (no dentro de cruge!!)
class MiClaseCrugeMailer extends CrugeMailerBase {

    public function sendEmail($to,$subject,$body){
        // usa esto para que el correo se envíe por la via estandar mail()
        // si quieres usar otro método deberás comentar esta linea e implementar
```

```

    // tu propio mecanismo.
    parent::sendEmail($to,$subject,$body);
}
}

```

2. En el "config/main" del componente 'crugemailer' pones el argumento:

```
'class' => 'application.components.MiClaseCrugemailer'
```

3. Esta nueva clase contendría un nuevo método llamado "notificarStockMercancia": Notarás que uso una llamada a `$this->render`, es éste método el que renderiza tu correo dentro de una *vista que has creado previamente*.

```

<?php
// esta clase va en: 'protected/components/MiClaseCrugemailer.php' (no dentro de cruge!!)
class MiClaseCrugemailer extends CrugemailerBase {

    public function notificarStockMercancia($usuario,$asunto) {
        // IMPORTANTE: NO implementes aqui el metodo de envio de email
        // por ejemplo mail(..) o phpmailer(..bla..) eso es
        // responsabilidad del metodo sendEmail que haces mas abajo
        // aqui solo dedicate a preparar el correo !!
        $this->sendEmail(
            $usuario->email,$asunto,
            $this->render('vista_stockvence',array('data'=>$usuario))
        );
    }

    public function sendEmail($to,$subject,$body){
        // usa esto para que el correo se envíe por la via estandar mail()
        // si quieres usar otro método deberás comentar esta linea e implementar
        // tu propio mecanismo.
        parent::sendEmail($to,$subject,$body);
    }
}

```

4. Para usar este nuevo método en tu aplicación:

```
Yii::app()->crugemailer->notificarStockMercancia($usuarioSeleccionado,"asunto");
```

¿ Cómo cambiar el método de envío de correo ? (mail, phpmailer etc)

Tienes dos alternativas para cambiar el método de envío de correo, supongamos que no te gusta o no te sirve la función `mail()` y prefieres usar la librería `PHPMailer`:

- Método 1: (preferido)

Editas tu propia clase 'MiClaseCrugemailer' que has creado en tu propia ruta de 'applications.components' y en ella editas el método sendEmail para implementar la solución.

- Método 2:

Editas CrugemailerBase.php y en ella editas el método sendEmail para implementar la solución.

Personalmente recomiendo el "método 1" ya que evita la manipulación del core de Crugemailer, delegando así a tu aplicación web la "especialización", además ayuda a que recibas actualizaciones a crugemailer via git sin que haya conflicto. El "método 2" es referenciado en el README pero para el momento de escribirlo no se había tomado en cuenta la actualización via GIT entre otras cosas.

Beneficios obtenidos al usar Crugemailer

- Se logra que tú hagas envíos de correos orientados a tu modelo de negocio, sin hacer espagueti con el código!!, centralizando y encapsulando el código... todos tus correos invocarán a sendEmail...y no tendrás que crear y duplicar o triplicar el esfuerzo de escribir las funciones de correo.
- Se logra que puedas clarificar el código...envías un correo con un propósito muy definido:

```
Yii::app()->crugemailer->nuevoCorreoAlCliente($usuario, "hola")
```

- Se logra que en cualquier momento puedas decidir por cual vía se envía el correo, si por mail() o por phpmailer() o por cualquier otra que tú decidas..solo cambiando un método..sin hacer "código espagueti" ("código espagueti" = código con hiper-dependencias, necio y ofuscado que termina acabando con la vida del proyecto entero en menos de tres meses gracias a la flojera de no organizar ni leer, ahh lo olvidaba, además en algo menos de 6 meses solo dios y tú (si te acuerdas) entenderán que se hizo ahí).

¿Cómo hacer un CRON (BATCH) con Crugemailer ?

Primero, qué es un "CRON". Es acrónimo para "CRON JOB" o "TAREA PROGRAMADA", que se encarga de despachar correos en forma de LOTES

(BATCH).

Segundo, CrugeMailer no va a correr como una tarea programada por si solo porque en PHP no existen las tareas programadas dado que la arquitectura de PHP no admite hilos secundarios como lo haría Java. Por tanto para hacer una tarea programada en PHP se invoca un script (o una funcion) cada cierto tiempo usando un TRIGGER (disparador):

Tipos de Disparador (TRIGGER):

Hay dos metodos para ejecutar una tarea programada en Php:

1) Con un CRONJOB de tu "sistema operativo" mandas a ejecutar un script periódicamente. Pregunta si tu hosting dispone de CronJobs, algunos lo tienen, en la config del hosting se le indica el tiempo y la ruta de un script y sera el servidor quien corra el script cada cierto tiempo. Si es windows, puedes crear una tarea programada que corra un .BAT con una sentencia PHP que ejecute el script, o, simplemente que corra un script por su URL.

2) Pones un contador (en una tabla o en memoria de sesion) en tu aplicación, y cada vez que alguien invoque a tu archivo "index.php" incrementas ese valor, y si se llega al "umbral" ejecutas un método en tu aplicación.

Sea cual sea el método de lanzamiento o "TRIGGER", siempre invocarás un método en una clase, el cual va a ejecutar una lectura de mensajes almacenados y los enviará por correo. Este es el concepto principal.

¿ Cómo hacerlo con CrugeMailer ?

(para saber mas lee "Cómo Personalizar CrugeMailer" mas arriba).

1. Primero, no mandas el email cada vez que ocurre algo en tu negocio (como normalmente sucedería sin un proceso de batch). Es decir, en vez de enviar el email de una vez lo almacenarás.

```
<?php
// esta clase va en: 'protected/components/MiClaseCrugeMailer.php' (no dentro de cruge!!)
class MiClaseCrugeMailer extends CrugeMailerBase {

    public function sendEmail($to,$subject,$body){
        // en vez de enviarlo:
        // parent::sendEmail($to,$subject,$body);

        // lo guardas en un modelo llamado protected/models/Correo.php
        $c = new Correo();
        $c->asunto = $subject;
        $c->body = $body; // es un LONGBLOB en tu modelo
```

```

        $c->to = $to;
        $c->enviado = false;
        $c->insert();
    }
}

```

2. Creas un metodo trigger() para que sea invocado mas adelante por el disparador del cronjob al momento que quieras enviar los correos realmente.

```

<?php
// esta clase va en: 'protected/components/MiClaseCrugemailer.php' (no dentro de cruge!!)
class MiClaseCrugemailer extends CrugemailerBase {

    // en config/main en el setup de crugemailer le pones:
    // 'limit'=>10, para indicar cuantos emailas por vez quieres mandar.
    public $limit;

    // metodo usado por tu disparador:
    // Yii::app()->crugemailer->trigger();
    //
    public function trigger() {
        $lista = Correo::model()->findAllByAttributes(array('enviado'=>false));
        $n=0;
        foreach($lista as $c){
            $this->sendEmailReal($c->to, $c->asunto,$c->body);
            if($n++ > $this->limit)
                break;
        }
    }

    public function sendEmailReal($to,$subject,$body){
        // AQUI PONES TU PHPMAILER...0..
        // simplemente por defecto a:
        // (este es el sendEmail de CrugemailerBase)
        parent::sendEmail($to,$subject,$body);
    }

    public function sendEmail($to,$subject,$body){
        // en vez de enviarlo:
        // parent::sendEmail($to,$subject,$body);

        // lo guardas en un modelo llamado protected/models/Correo.php
        $c = new Correo();
        $c->asunto = $subject;
        $c->body = $body; // es un LONGBLOB en tu modelo
        $c->to = $to;
        $c->enviado = false;
        $c->insert();
    }
}

```

3. Ahora creas un script disparador. (IMPORTANTE: considera protegerlo de ejecución desde IP's extrañas)

por ejemplo asumo el metodo1 de ejecución (usando un script, ver inicio del tema) tu aplicación web tendrá un CronJob de sistema operativo que lanzará este script periodicamente:

```
http://tuapp.com/cronjob-email.php
```

```
<?php
// llámalo: /tuapp/cronjob-email.php
//
$yii=dirname(__FILE__).'/../../yii/framework/yii.php';
$config=dirname(__FILE__).'/protected/config/main.php';

defined('YII_DEBUG') or define('YII_DEBUG',false);
defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);

$_GET['r'] = '/site/disparador';

require_once($yii);
Yii::createWebApplication($config)->run();
```

4. En siteController.php, creas un action PROTEGIDO por un CAccessControl Rule de Yii que prevenga que una IP desconocida lo ejecute: (mira la documentación yii de CAccessControl)

```
public function actionDisparador(){
    $this->crugemailer->trigger();
}
```

LISTO. tienes envío de correos en procesos de lotes usando un cron-job.

CASO REAL: Crea tu propia clase MiCrugemailer

Aquí tienes un caso real de uso de Crugemailer. Ir al enlace

ENCRIPTADO DE CLAVES

Por defecto Cruge trae la encriptación de claves desactivada. Esto es así para facilitar el trabajo mientras se instala. Igualmente puedes cambiar el método de encriptación de claves por el de tu preferencia.

CÓMO ACTIVAR LA ENCRIPTACION

Para activar o desactivar la encriptación se usa el atributo 'useEncryptedPassword' => true o false (en tuapp/protected/config/main.php)

```
'cruge'=>array(
...
'useEncryptedPassword' => true,
...
),
```

CÓMO SELECCIONAR EL METODO DE ENCRIPCIÓN

Se usa el atributo 'hash' => 'md5', (en tuapp/protected/config/main.php). Los valores aceptados por el parámetro hash son cualquiera de los aceptados por la función "hash" (PHP 5.1+), para mayor información debes visitar: <http://www.php.net/manual/en/function.hash-algos.php> hash_algos() (<http://www.php.net/manual/en/function.hash-algos.php>). (Gracias a Ricardo Andrés Obregón).

```
'cruge'=>array(
...
'hash' => 'md5',
...
),
```

COMO ENCRIPAR LA CLAVE DE ADMIN

Esto es requerido cuando pasas a producción. Ya que la clave de admin es humanamente visible para efectos de desarrollo. Una vez que selecciones el algoritmo de encriptación usando el atributo hash de config/main entonces deberás encriptar manualmente la clave de admin en tu base de datos.

1. Crea un script PHP que usaras para generar la nueva clave basado en el algoritmo de encriptacion seleccionado. Si el atributo de config/main 'hash' dice 'md5' entonces podrias hacer esto:

```
-- inicio archivo: generador-clave.php --
<?php // uso: http://cosa.com/generador-clave.php?clave=minuevaclave
echo "Clave encriptada es: [" .md5($_GET['clave'])."]";
?>
```



```
!-- fin archivo: generador-clave.php --
```

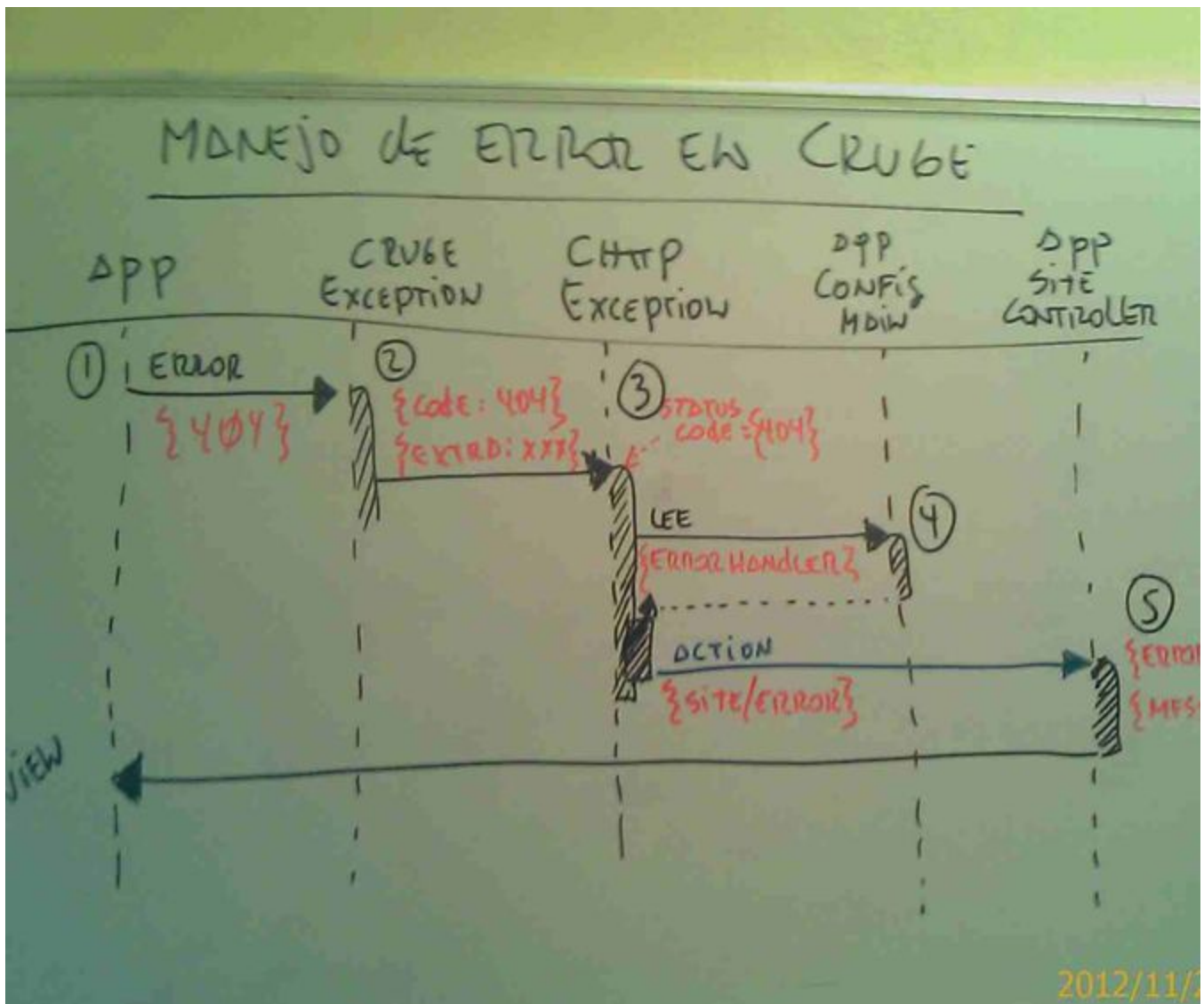
2. La clave obtenida con el script `generador-clave.php` la guardarás manualmente via SQL usando phpMyAdmin o similar administrador de base de datos. Editarás el campo 'password' de la tabla 'cruge_user' para el registro con username: 'admin'.

MANEJO DE ERRORES (EXCEPCIONES) EN CRUGE

Cruge emite una excepción mediante el componente `CrugeException` (<https://bitbucket.org/christiansalazarh/cruge/src/e1d6c54c7816f8127d5a1ce26bf2d0a5a4889a31/components/CrugeException.php?at=master>), que es una extensión de `CHttpException` (<http://www.yiiframework.com/doc/api/1.1/CHttpException>)

Además de estas dos clases hay un tercer componente involucrado: el `config/main` en la sección de 'errorHandler' (<https://bitbucket.org/christiansalazarh/crugeholamundo/src/875739385a55/protected/config/main.php?at=master#cl-138>) en el cual le das la ruta del action (<https://bitbucket.org/christiansalazarh/crugeholamundo/src/875739385a55/protected/controllers/SiteController.php?at=master#cl-35>) que va a responder ante una excepción.

Este diagrama muestra cómo interactúan las clases y partes del sistema ante una excepción.



- 1) cruge emite una excepcion con CrugeException, ejemplo 501 access denied.
- 2) CrugeException deriva de CHttpException y le pasa sus argumentos en el constructor.
- 3) CHttpException se inicializa
- 4) CHttpException lee la config/main y sabe a que action debe redirigirse.
- 5) CHttpException redirige el browser al action indicado.
- 6) (fin) se ha renderizado la vista site/error en la aplicacion.

ERRORES FRECUENTES

Excepcion: no se pudo hallar el sistema de configuracion, quiza la tabla cruge_system esta vacia o ha indicado un identificador de sistema inexistente

Este error es generado cuando se ha instalado cruge pero la tabla cruge_system no tienen ningún dato. Por defecto cuando el script sql se instala por primera vez este trae datos para cruge_system. Cruge_system es una fila que informa acerca de las variables de un sistema, pueden crearse varias configuraciones a las cuales se les hace referencia por su nombre, si en config/main hay una referencia a esta configuración y esta no existe en cruge_system entonces este error aparecerá.

Excepción: por favor cambie las referencias a 'useridentity' por 'crugeuser'

Cuando ocurre ? cuando ejecutas: `http://localhost/tuapp/index.php?r=site/login`, llenas el form de login y le das clic a Aceptar. Eso causará la excepción. **Por qué ocurre ?** Esto sucede principalmente porque estás usando el formulario de login por defecto de Yii, en cambio debes usar el de Cruge, al cual se accede mediante: `<?php echo Yii::app()->user->ui->loginLink; ?>`

Alias

"application.modules.cruge.components.CrugeWe is invalid. Make sure it points to an existing PHP file and the file is readable.

Por favor revisa este issue: ver el issue 23 el cual da la solución (<https://bitbucket.org/christiansalazarh/cruge/issue/23/alias>)

The authenticity of host 'bitbucket.org (207.223.240.182)' can't be established.

Lee el siguiente issue. Ver solución aquí (<https://bitbucket.org/christiansalazarh/cruge/issue/24/error-al-hacer-clone-del-repositorio-cruge>), esto se debe a que no tienes una clave SSH y estás usando git clone en modo ssh, debes usar git clone en modo http.

Esta página web tiene un bucle de redireccionamiento. Error 310 (net::ERR_TOO_MANY_REDIRECTS): Demasiados redireccionamientos

CAUSA PRINCIPAL: Esto es porque has eliminado todos los permisos al usuario Invitado, quizá eliminando al usuario Invitado, o denegando explícitamente el

acceso al invitado al siteController o al controller encargado de gestionar errores.

CAUSA ALTERNA: También puede darse si rediriges a un usuario a un action al cual no tiene permiso haciendolo desde otro action que tampoco tiene permiso debido a sesion caducada, causando bucle.

CONDICIONES: 1. Cuando ocurra algun caso en que usuario acceda a un action no autorizado. 2. Cuando la sesion haya caducado.

SOLUCION: Si el caso es relativo al invitado, entonces dale al invitado permisos para acceder a "siteController" (no a los actions, ya que Cruge se autoprotege y siempre concede acceso a actionError de siteController, pero si siteController esta bloqueada por ti entonces no podra entrar causandose un bucle).

ayuda: usa los recursos de configuración de Cruge: "allowUserAlways" en conjunto con esta linea al pie de tu layout para saber que errores ocurrieron:

```
<?php echo Yii::app()->user->ui->displayErrorConsole(); ?>
```

lee acerca de Cruge#El_usuario_Invitado y de Cruge#Modo_de_Programaci.C3.B3n_del_RBAC

I18N

Si ves, Cruge escribe los mensajes en español, mientras que su código esta en inglés. Todos los mensajes se dirigen a la clase CrugeTranslator::t("mensaje en español"), por tanto ese es el punto para traducir a otro idioma. En un futuro nuevo commit se harán traducciones. Pronto.

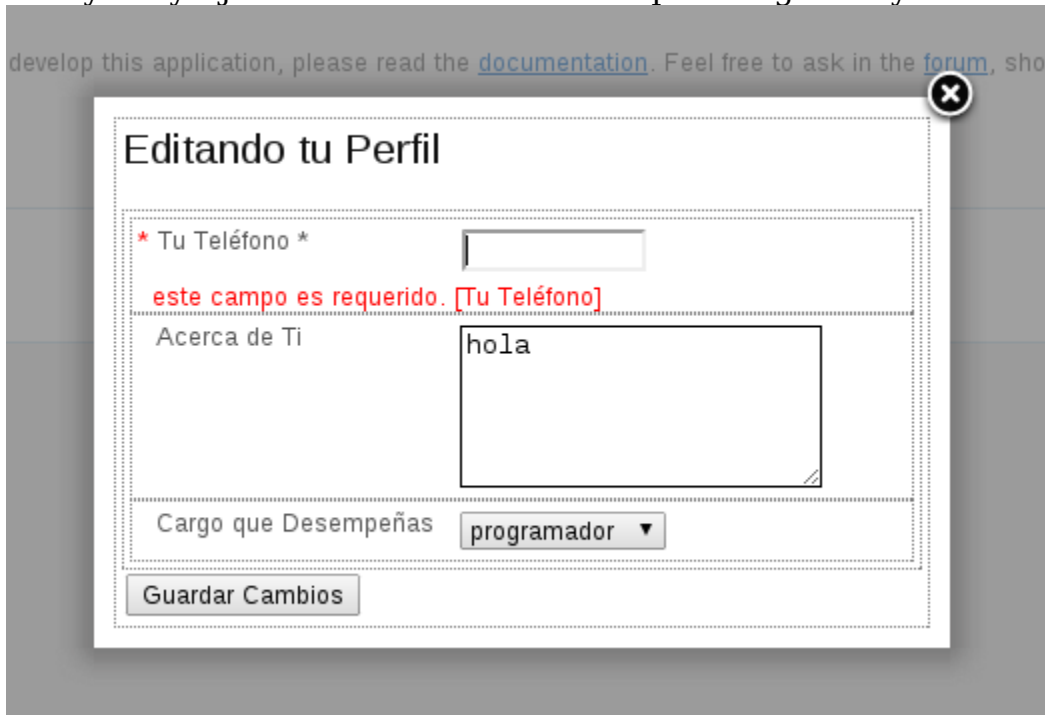
CASOS DE EJEMPLO

Este apartado es para presentar casos de ejemplo en donde pudieras pensar que Cruge requiere modificaciones, pero en realidad el problema puede solucionarse de otra manera.

igualmente puedes contribuir con tus propios casos enviandomelos al correo christiansalazarh@gmail.com, normalmente yo saco los casos de ejemplo de las preguntas enviadas por correo o de los issues.

Cruge con Fancybox y Ajax

Edita el perfil de tu usuario con sus campos personalizados de Cruge usando Fancybox y Ajax. mas documentacion aqui: CrugeFancybox.



Tu propia vista de edicion de perfil de usuario

No es algo que no conozcas de Yii y el uso de formularios estandar. Es lo mismo, salvo uno que otro uso del API de Cruge para obtener cierta informacion clave. Es practicamente un copia-pegas de aquella vista que Cruge te ofrece por defecto.

(necesitas una para el Login o Registration ? haz lo mismo que se hizo con Profile, simplemente buscas en modules/cruge/ui la vista requerida y copias y pegas)

1) en un controller tuyo (MyController), creas un action: actionProfile

```

/*
 array('MyController/profile', 'id'=>'the unique id')
*/
public function actionProfile() {
    $model = Yii::app()->user->user; // ciudadano es: user->user, el cual da al CrugeStoredUser
    Yii::app()->user->um->loadUserFields($model); // le pedimos al api que cargue los campos personal
    $this->performAjaxValidation('crugestoreduser-form', $model);
    $postName = CrugeUtil::config()->postNameMappings['CrugeStoredUser'];
    if (isset($_POST[$postName])) {
        $model->attributes = $_POST[$postName];
        if ($model->validate()) {
            $newPwd = trim($model->newPassword);
            if ($newPwd != '') {

```

```

        Yii::app()->user->um->changePassword($model, $newPwd);
        Yii::app()->crugemailer->sendPasswordTo($model, $newPwd);
    }
    if (Yii::app()->user->um->save($model, 'update')) {
        Yii::app()->user->setFlash('profile-flash',
            'Your Profile has been saved.');
```

2) Creas la vista 'profile', es practicamente un copia-pega de aquella vista que Cruge te ofrece por defecto, a excepcion del flash que he agregado aqui. esta vista esta almacenada en: 'protected/views/mycontroller/profile.php',

```

<?php
// $model: algo que implementa a ICrugeStoredUser,
?>
<?php if(Yii::app()->user->hasFlash('profile-flash')): ?>
<div class="flash-success">
    <?php echo Yii::app()->user->getFlash('profile-flash'); ?>
</div>
<?php else: ?>
<div class="form" id='profile-form' >
<?php $form = $this->beginWidget('CActiveForm', array(
    'id'=>'crugestoreduser-form',
    'enableAjaxValidation'=>false,
    'enableClientValidation'=>false,
)); ?>
<div class="row form-group">
    <div class='field-group'>
        <div class='col'>
            <?php echo $form->labelEx($model,'username'); ?>
            <?php echo $form->textField($model,'username'); ?>
            <?php echo $form->error($model,'username'); ?>
        </div>
        <div class='col'>
            <?php echo $form->labelEx($model,'email'); ?>
            <?php echo $form->textField($model,'email'); ?>
            <?php echo $form->error($model,'email'); ?>
        </div>
        <div class='col'>
            <?php echo $form->labelEx($model,'newPassword'); ?>
            <?php echo $form->textField($model,'newPassword',array('size'=>10)); ?>
            <?php echo $form->error($model,'newPassword'); ?>
            <script>
                function fnSuccess(data){
                    $('#CrugeStoredUser_newPassword').val(data);
                }
                function fnError(e){
                    alert("error: "+e.responseText);
                }
            </script>
            <?php echo CHtml::ajaxbutton("Generate a new Password"
                ,Yii::app()->user->ui->ajaxGenerateNewPasswordUrl
                ,array('success'=>new CJavaScriptExpression('fnSuccess'),
                    'error'=>new CJavaScriptExpression('fnError'))
```

```

        ); ?>
    </div>
</div>
<?php
    if(count($model->getFields()) > 0){
        echo "<div class='row form-group'>";
        foreach($model->getFields() as $f){
            //if(($f->fieldname == 'picture') || ($f->fieldname == 'bigpicture')) continue;
            echo "<div class='col' id='col_{$f->fieldname}'>";
            echo Yii::app()->user->um->getLabelField($f);
            echo Yii::app()->user->um->getInputField($model,$f);
            echo $form->error($model,$f->fieldname);
            echo "</div>";
        }
        echo "</div>";
    }
?>
<div class="row buttons">
    <?php Yii::app()->user->ui->tbutton("Save Profile"); ?>
</div>
<?php echo $form->errorSummary($model); ?>
<?php $this->endWidget(); ?>
</div>
<?php endif; ?>

```

Acceder a los campos Personalizados desde tu propia clase relacionada con CrugeUser

Supon que tienes una clase llamada Empleado, que esta asociada a un CrugeStoredUser (un iduser de cruge, aquel id que sacas de `Yii::app()->user->id`).

Ahora supon que te gustaria que este modelo empleado leyera los campos personalizados del usuario de cruge al cual esta asociado, algo como:

```

    $e = Empleado::model()->findByPk(123); // levantas al empleado con idempleado = 123.
    echo "Cédula": $e->cedula;
    echo "Nombre": $e->nombre;
    echo "Apellido": $e->apellido;

```

El atributo "cedula", "nombre" y "apellido" son campos personalizados que hiciste con el gestor de campos personalizados de Cruge, y , mágicamente pueden ser vistos publicamente como si fuesen parte de la clase, gracias a los magic getters and setters de PHP (no de yii).

```

<?php
class Empleado extends CActiveRecord {

```

```

    public function __get($name){
        $field = Yii::app()->user->um->loadFieldByName($name);
        if($field != null)
            return Yii::app()->user->um->getFieldValue($this->iduser0,$field);
        return parent::__get($name);
    }
    public function __set($name,$val){
        $field = Yii::app()->user->um->loadFieldByName($name);
        if($field != null)
            return;
        return parent::__set($name,$val);
    }
    public function relations()
    {
        return array(
            'iduser0' => array(self::BELONGS_TO, 'CrugeStoredUser', 'iduser'),
        );
    }
}

```

Presentar campos personalizados de CrugeUser en un CGridView

Supon que tienes un modelo llamado Empleado, el cual referencia a un CrugeStoredUser mediante Empleado::iduser.

Ejemplo:

```

/*
 @param string $idempleado (primarykey)
 @param string $cargo (texto)
 @param string $iduser (apunta a un CrugeStoredUser)
*/
class Empleado extends CActiveRecord {
    // ...una clase normalmente creada con GII para este caso...
}

```

ahora supon que quieres listar esos empleados en un CGridView:

```

$this->widget('zii.widgets.grid.CGridView', array(
    'dataProvider'=>$dataProvider,
    'columns'=>array(
        array('name'=>'idempleado'),
        array('name'=>'iduser'),
        array('name'=>'cargo'),
        array(
            'class'=>'CButtonColumn',
        ),
    ),
)

```



```

),
));

```

asi tan simple mostrara:

```

idempleado iduser cargo
-----
0000000011 001 administrador
0000000012 002 jefe
0000000013 003 supervisor

```

Ahora...en vez de \$iduser quieres que salga el "nombre y apellido" del usuario, ambos campos personalizados en Cruge, pues debes hacer estos cambios:

```

/*
 * @param string $idempleado (primarykey)
 * @param string $cargo (texto)
 * @param string $iduser (apunta a un CrugeStoredUser)
 */
class Empleado extends CActiveRecord {
    // ...una clase normalmente creada con GII para este caso...

    private function _getCrugeUser(){
        return Yii::app()->user->um->loadUserById(
            $this->userid,true);
        // TRUE para que cargue los campos personalizados
    }

    public function getNombre(){
        return Yii::app()->user->um->getFieldValue(
            $this->_getCrugeUser(),'nombre');
    }

    public function getApellido(){
        return Yii::app()->user->um->getFieldValue(
            $this->_getCrugeUser(),'apellido');
    }
}

```

y finalmente en el CGridView haces estos cambios:

```

$this->widget('zii.widgets.grid.CGridView', array(
    'dataProvider'=>$dataProvider,
    'columns'=>array(
        array('name'=>'idempleado'),
        array('name'=>'iduser'),
        array('name'=>'nombre'),
        array('name'=>'apellido'),
        array('name'=>'cargo'),
    )
);

```

```
        array(  
            'class'=>'CButtonColumn',  
        ),  
    ),  
);
```

asi tan simple mostrara:

```
idempleado  iduser  nombre  apellido  cargo  
-----  
'0000000011  001    christian salazar  administrador  
'0000000012  002    juan     perez    jefe  
'0000000013  003    pedro   jose     supervisor
```

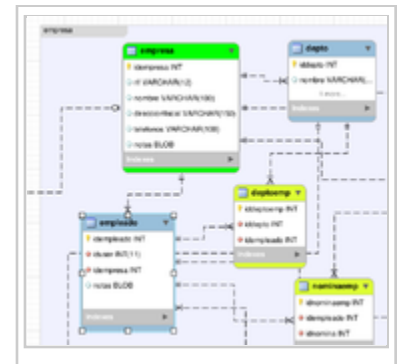
El empleado debe pertenecer a una o varias empresas

Lecturas recomendadas:

1. Clases y Tablas, Por qué no son lo mismo (<http://trucosdeprogramacionmovil.blogspot.com/2013/02/clases-y-tablas-por-que-no-son-lo-mismo.html>)
2. Modelando con Cruge y la libreria EYUI

Pudieras pensar que la tabla de usuarios de cruge (cruge_user) requiere mas campos, por ejemplo: idempresa. Pues no es asi. Puedes ampliar la imagen a la derecha para ver como insertar cruge en una relacion mas amplia.

Los campos personalizados de Cruge NO SON para modelar tu sistema, es decir, no son para conectar a Cruge con las reglas de negocio de tu cliente, son para que el usuario introduzca su propia información de perfil, nada mas.



Cuando necesitas que tu usuario de Cruge "tenga mas atributos" es porque lo que esta sucediendo es que tu modelado se esta desarrollando. En consecuencia no debes hacer recaer tu modelado sobre el usuario de Cruge, es un error hacerlo: el usuario de Cruge solo representa a la persona real que esta usando tu sistema.

Si por ejemplo, ese usuario de Cruge es el Empleado de una Empresa pues entonces LA EMPRESA y EL EMPLEADO DEBEN EXISTIR COMO ENTIDADES SIN INTENTAR RETORCER EL USUARIO DE CRUGE PARA HACERLO CUMPLIR

ESAS NECESIDADES.

La Empresa, El Empleado y El Usuario (de Cruge) son entidades separadas, existen en la realidad, son sustantivos. En modelado UML el sustantivo se representa como una "Clase" en donde cada objeto de esa clase (la instancia de la clase) se almacena en la mal llamada y famosa "tabla" (que se llama "relacion" o "modelo entidad-relacion", de este concepto práctico es que nacen estos nombres, no del "caletre memorizado" de los libros que aquellos sabios de la nasa pretenden lucir sin saber el porqué de su existencia).

Entendiendo el parrafo anterior comprenderás que UNA PERSONA (representada por el usuario de Cruge) puede laborar en Varias Empresas (Es Empleado de) y que una Empresa tendrá Varios Empleados. La Empresa y el Usuario de Cruge son las entidades, los sustantivos, y el Empleado es "Una Relación, una Clase de Asociación" entre dichas entidades. Esto es clave que lo comprendas.

La clase EMPLEADO "DEPENDE DE" EMPRESA y "DEPENDE DE" CrugeUser:

[CRUGEUSER]<----[EMPLEADO]----->[EMPRESA] (flecha punteada ----> significa: "depende de")

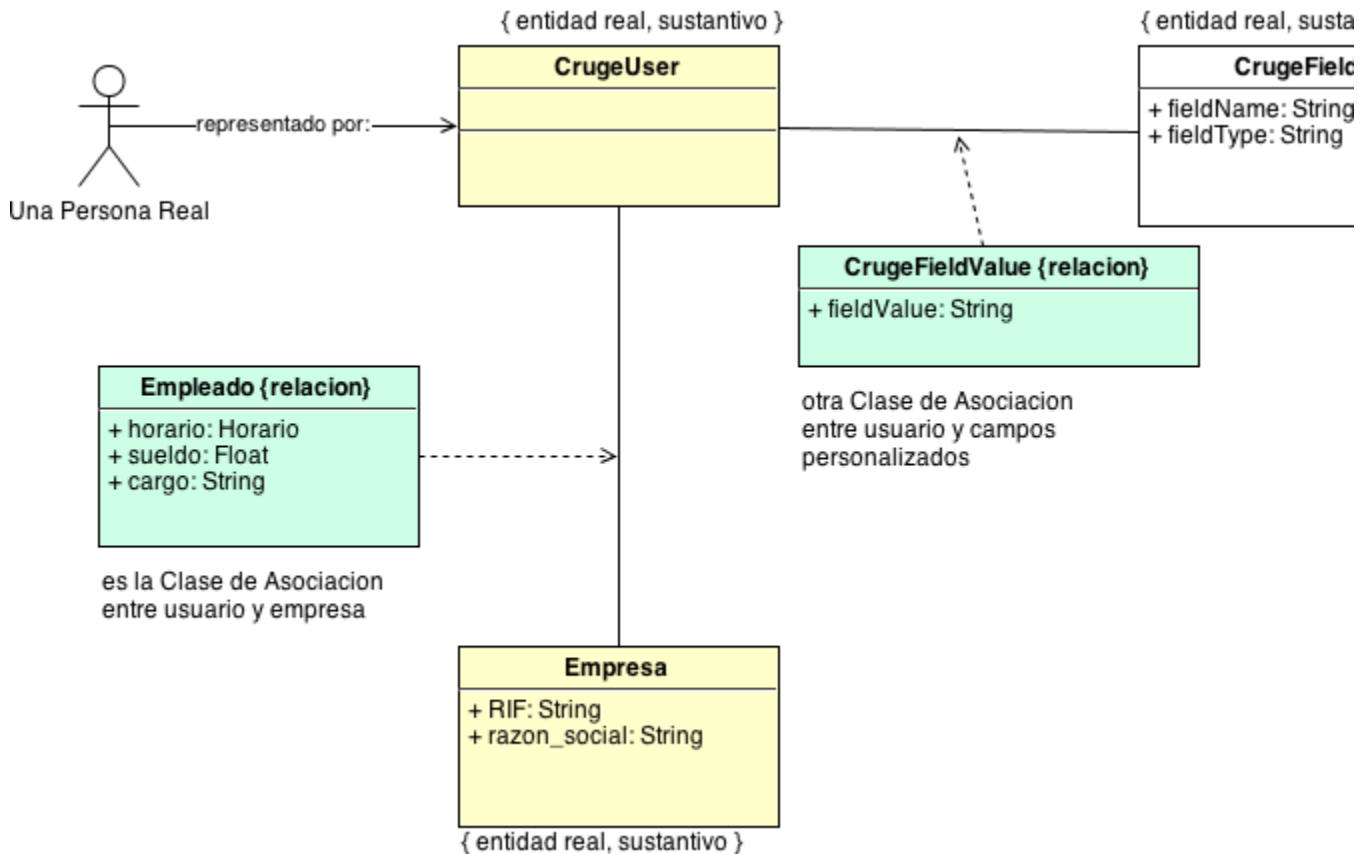
La clase EMPLEADO es una relacion XY porque REPRESENTA LA RELACION entre un USUARIO de cruge y una EMPRESA, en UML La clase EMPLEADO se conoce como "una clase de asociación" porque representa la asociación que existe entre dos entidades. Esta clase EMPLEADO puede tener atributos propios de esa relacion: "sueldo_asignado", "horario_asignado", "cargo_que_ejecuta". (estabas pensando meter toda esta basura en los campos personalizados de Cruge ?..si verdad ?).

La escuela de informática formal respecto a modelado de datos indica que cuando hay una relacion "1:1" esta siempre es absorbida por una de las partes por razones de optimización y ahorro de espacio, desapareciendo entonces esa relación. En otras palabras, si Juan Perez (representado por el usuario de Cruge) labora en SOLO UNA EMPRESA por ley fuerte de negocio, entonces tendrás una "RELACION 1:1" para la clase EMPLEADO contra CrugeUser la cual desaparecerá (Empleado) quedando sus atributos delegados a CrugeUser.

Por qué a veces es sano desaparecer a esa relación 1:1 ? Cuando el volumen de datos así lo pide. Si hay un millón de empleados, tendrás entonces un millon de objetos almacenados de clase CrugeUser, mas un millón de objetos almacenados de clase Empleado. Es en estos casos es donde entra en juego la optimización, ya que al absorber la relación Empleado dentro de CrugeUser entonces ahorrarás ESPACIO (no velocidad, la cual te la dará el buen diseño de índices).

En el caso de que efectivamente se haga una absorción 1:1 sí requerirás de campos personalizados extra en CrugeUser para indicar los nuevos atributos que vienen de la extinta clase Empleado, pero...esta "absorción 1:1" al modelado UML

le sabe a pastel de gato, aqui en UML lo que tiene validez y prioridad sobre la comodidad del desarrollador y el ahorro de espacio es "el modelado": Si el modelado dice que teóricamente es: CrugeUser--->Empleado--->Empresa entonces *será así* por fuerza natural, será comprensible, entendible a futuro, dejando atrás la idea a veces super generalizada de querer optimizar lo que no se debe optimizar so pena de ahogar el modelado, si te gusta "optimizar" optimiza tu análisis de modelado y eso si será una optimización el resto es basura.



El caso real quedaría así como muestro a continuación, este ejemplo contiene lo que quiero mostrar: relations(), pero aparte contiene también como se manejan campos personalizados en el modelo Empleado el cual es asociado con CrugeStoredUser.

```

<?php
/**
 * This is the model class for table "empleado".
 *
 * The followings are the available columns in table 'empleado':
 * @property integer $idempleado
 * @property integer $iduser
 * @property integer $idempresa
 * @property string $notas
 */
  
```

```

class Empleado extends CActiveRecord
{
    public function getPresentacion(){
        // usa los magic getters abajo para leer campos personalizados
        return ucwords($this->nombre." ".$this->apellido);
    }
    /**
     * devuelve la lista de empresas en la cual labora el usuario que ha iniciado sesion
     */
    public static function misEmpresas(){
        $list = self::model()->findAllByAttributes(array('iduser'=>Yii::app()->user->id));
        $result = array();
        if(isset($list))
            foreach($list as $empleadoInst)
                $result[] = $empleadoInst->idempresa0;
        return $result;
    }
    public static function isEmpleadoEmpresa($idempresa){
        foreach(self::misEmpresas() as $emp)
            if($emp->primaryKey == $idempresa)
                return $emp;
        return null;
    }
    /**
     * retorna la instancia de empleado del usuario indicado en la empresa seleccionada
     */
    public static function getEmpleado($iduser,$idempresa){
        return Empleado::model()->findByAttributes(array('iduser'=>$iduser,'idempresa'=>$idempres
    }
    public function getEmpresa(){
        return $this->idempresa0;
    }
    /**
     * getEmpleadoPorCedula
     * busca a un empleado por su cedula, considerando que la cedula
     * es un campo personalizado del sub sistema Cruge.
     * @param string $cedula
     * @param integer $idempresa
     * @static
     * @access public
     * @return instancia de Empleado o nul
     */
    public static function getEmpleadoPorCedula($cedula,$idempresa){
        $user = Yii::app()->user->um->loadUserByCustomField(
            'cedula', $cedula);
        if($user != null){
            return self::getEmpleado($user->primaryKey, $idempresa);
        }
        else
            return null; // ningun usuario del sistema tiene esta cedula
    }
    /**
     * este getter permite que sobre el modelo empleado se hagan consultas de atributo
     * sobre los campos personalizados, ejemplo:
     *
     * echo $empleado->cedula; ,siendo cedula un campo personalizado del sistema Cruge
     */
    public function __get($name){
        $field = Yii::app()->user->um->loadFieldByName($name);
        if($field != null)
            return Yii::app()->user->um->getFieldValue($this->iduser0,$field);
        return parent::__get($name);
    }
    public function __set($name,$val){

```

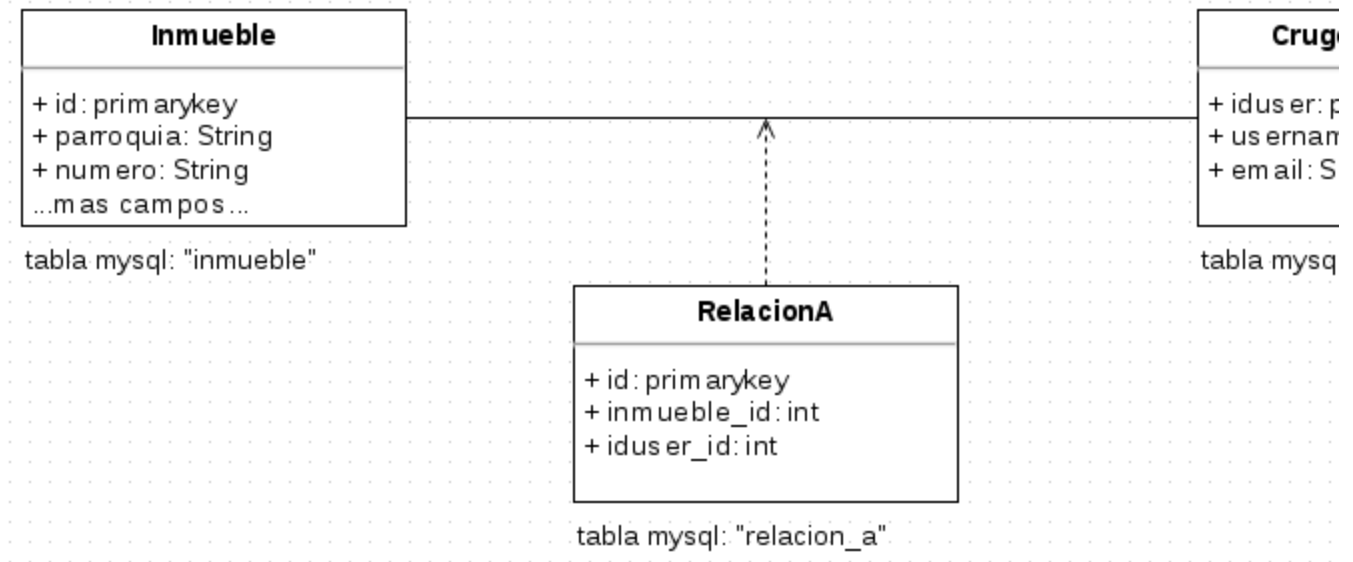
```
        $field = Yii::app()->user->um->loadFieldByName($name);
        if($field != null)
            return;
        return parent::__set($name,$val);
    }
    /**
     * @return array relational rules.
     */
    public function relations()
    {
        // NOTE: you may need to adjust the relation name and the related
        // class name for the relations automatically generated below.
        return array(
            'deptoemps' => array(self::HAS_MANY, 'Deptoemp', 'idempleado'),
            'iduser0' => array(self::BELONGS_TO, 'CrugeStoredUser', 'iduser'),
            'idempresa0' => array(self::BELONGS_TO, 'Empresa', 'idempresa'),
            'nominaemps' => array(self::HAS_MANY, 'Nominaemp', 'idempleado'),
            'puestoemps' => array(self::HAS_MANY, 'Puestoemp', 'idempleado'),
        );
    }
}
```

Mostrando Campos Personalizados en una Relacion

Este ejemplo lo he creado en base a una pregunta realizada por el desarrollador xxxxx, me pareció que es un caso común entre los desarrolladores que usan Cruge, por esta razón publicaré la solución aquí.

la pregunta original fue:

"..mi relación viene dada con una tabla que se llama 'inmueble' que conecta con una tabla de nombre 'relacion_a' y a su vez acá hago la relación con la tabla de 'cruge_user'. Como bien te dije quiero una relación de muchos a muchos, para lo que he usado la siguiente sentencia en mi modelo Inmueble: " (ver el diagrama)



La solución rápida y cruda:

La solución Cruda, simple, sin profundizar mucho en modelado que le propongo al desarrollador es crear una vista con el siguiente contenido:

```

<h1>Inmuebles Relacionados con Usuarios</h1>
<?php
$model = new RelacionA('search');
$this->widget('zii.widgets.grid.CGridView', array(
    'dataProvider'=>$model->search(),
    // 'rowHtmlOptionsExpression' es usada para cargar campos personalizados
    'rowHtmlOptionsExpression'=>'$data->user0->getUserDescription(true)',
    'columns'=>array(
        // los datos de la relacion_XY (en este caso RelacionA)
        //
        array('name'=>'inmueble_id'),
        // los datos de la clase relacionada Inmueble
        //
        array('name'=>'inmueble0.parroquia'),
        array('name'=>'inmueble0.numero'),
        // los datos de CrugUser
        //
        array('name'=>'user0.email'),
        array('header'=>'First Name', 'value'=>'$data->user0->getCustomFieldValue(\'firstname\')'),
        array('header'=>'Last Name', 'value'=>'$data->user0->getCustomFieldValue(\'lastname\')'),
    ),
));
?>
  
```

Previamente, para que esa vista funcione será necesario asegurar dos cosas:

1. Que la tabla "relacion_a" tenga un INDICE PRIMARIO (crearle un campo ID autonumerico marcado como primary_key), sino el relacioador de Yii fallará.

2. Indicar las relaciones en el modelo RelacionA.php (modelo RelacionA.php que asumo ya habrás creado con la herramienta GII, lo mismo para el modelo Inmueble.php)

```
// archivo: RelacionA.php, CUIDADO: RelacionA necesita un indice primario definido!
public function relations()
{
    return array(
        'user0' => array(self::BELONGS_TO, 'CrugeStoredUser', 'iduser_id'),
        'inmueble0' => array(self::BELONGS_TO, 'Inmueble', 'inmueble_id'),
    );
}
```

Lo que producirá:

Inmuebles Relacionados con Usuarios

| Inmueble | Direccion | Numero | Correo | |
|----------|-----------|--------|---------------------|-------|
| 4 | dir1 | 1001 | username1@gmail.com | Chris |
| 5 | dir2 | 1002 | username1@gmail.com | Chris |
| 4 | dir1 | 1001 | anahi@prueba.com | Anah |
| 6 | dir3 | 1003 | anahi@prueba.com | Anah |
| 8 | dir5 | 1005 | anahi@prueba.com | Anah |

Alternativa que tambien va a funcionar:

```
<h2>otra manera, usando a getUserDescription</h2>
<?php
$this->widget('zii.widgets.grid.CGridView', array(
    'dataProvider'=>$model->search(),
    'columns'=>array(
        // los datos de la relacion_XY (en este caso RelacionA)
        //
        array('name'=>'inmueble_id'),
        // los datos de la clase relacionada Inmueble
        //
        array('name'=>'inmueble0.parroquia'),
        array('name'=>'inmueble0.numero'),

        // datos de CrugeUser
        //
        array('value'=>'$data->user0->getUserdescription(true)'),
    ),
));
```


Lo que producirá:

Otra manera, usando a getUserDescription

| Inmueble | Direccion | Numero | |
|----------|-----------|--------|------------------|
| 4 | dir1 | 1001 | Christian,Salaza |
| 5 | dir2 | 1002 | Christian,Salaza |
| 4 | dir1 | 1001 | Anahi,Silva |
| 6 | dir3 | 1003 | Anahi,Silva |
| 8 | dir5 | 1005 | Anahi,Silva |

Hasta aqui esta listo, se puede resolver el problema asi como esta, pero la solución optima es:

Usando Modelado para relacionar un modelo con CrugeStoredUser

Arriba, el primer diagrama es una clara representación de lo que llamo "Una Relacion XY" (Leer acerca de una relacion_xy).

Una Relacion XY es lo que llamo en terminos cortos a la representación de una clase de asociación, es decir: El Inmueble y el Usuario de Cruge se relacionan entre si mediante "la clase de asociación" llamada "RelacionA" (asi la llamo el desarrollador que propuso el ejemplo, pero que yo llamaria en cambio UsuarioInmueble).

Insertando Funcionalidad al modelo Inmueble

Este modelo "RelacionA" -no deberia ser usado- en una vista directamente, sino solo usado internamente por Inmueble porque a nivel de modelado "es" la relacion, la clase de asociacion que existe entre Inmueble y CrugeUser. En la solución cruda que he puesto arriba se esta usando, pero solo es para mostrar el punto, no deberia hacerse asi. Deberia ser parte implicita en el modelo "Inmueble", me refiero a insertar en Inmueble.php un metodo que "liste los usuarios asociados al inmueble":

```
class Inmueble extends CActiveRecord
{
    // lo principal: la relacion
    //
    public function relations()
    {
        return array(
            'relationship' => array(self::HAS_MANY, 'RelacionA', 'inmueble_id'),
        );
    }
}
```


NERD. Listando los inmuebles, y consultando la relacion via modelado.

| ID | Direccion | Numero | Asignac |
|----|-----------|--------|--------------------------------|
| 4 | dir1 | 1001 | Christian Salazar, Anahi Silva |
| 5 | dir2 | 1002 | Christian Salazar |
| 6 | dir3 | 1003 | Anahi Silva |
| 7 | dir4 | 1004 | |
| 8 | dir5 | 1005 | Anahi Silva |
| 9 | dir6 | 1006 | |
| 10 | dir7 | 1007 | |

mas adelante, podrias requerir en tu modelo listar o manipular a los usuarios asociados al inmueble, lo harias asi:

```

$model = Inmueble::model()->findByPk(4);
foreach($model->users as $u){
    echo "[asociado con usuario: ".$u->userdescription."]";
    // o tambien asi:
    // $u->getUserDescription(true);
    // echo "[asociado con usuario: ".$u->getCustomFieldValue('cedula')."]";
}

```

Para saber mas sobre getUserDescription, puedes consultar aqui:
<https://bitbucket.org/christiansalazarh/cruge/commits/ceb8b3bcfe3b69dceb07b18648181f558ffd67ec>

Personalizar la lista de valores de opcion de un Campo Personalizado tipo LISTBOX

"..al crear un campo personalizado que sea de tipo dropDown (listbox), se podría hacer que los items de la lista provengan del resultado de una consulta ? "

SI. haciendolo manualmente, por ahora en esta version de Cruge clonando la funcionalidad de Yii::app()->user->um->getInputField(\$model,\$f); en tu propia vista.

1. creas tu propia vista de edicion de perfil, copiando y pegando esto:
<https://bitbucket.org/christiansalazarh/cruge/src/b87379027c08/views/ui/usermanagementupdate.php?at=master> (esta demas decirlo, adaptandola a tu

UI)

2. toma nota, fijate donde dice: echo

Yii::app()->user->um->getInputField(\$model,\$f); es ahí donde se genera el componente de UI, es decir, un textbox, un listbox (tu caso). para el listbox, este es llenado mediante una tira de valores separados por coma, puedes verlo con mayor precisión en la línea 705 de: <https://bitbucket.org/christiansalazarh/cruge/src/b87379027c08/components/CrugeUserManager.php?at=master#cl-705>

3. entonces, en vez de usar a : echo

Yii::app()->user->um->getInputField(\$model,\$f); en cambio creas tu propio método getInputField, en un componente que tu hagas, y en cambio harías: echo Yii::app()->miPropioComponente->getInputField(\$model,\$f); y harías casi lo mismo, a excepción de la lista de valores de \$arOpt la cual vendría de tu propio modelo.

CREANDO MENU ITEMS BASADOS EN RBAC CON CRUGE

Cruge incorpora una función avanzada para crear menu items basados en las reglas rbac, este menu que se le presenta al usuario se genera automáticamente en base a la programación RBAC del usuario activo.

My Web Application

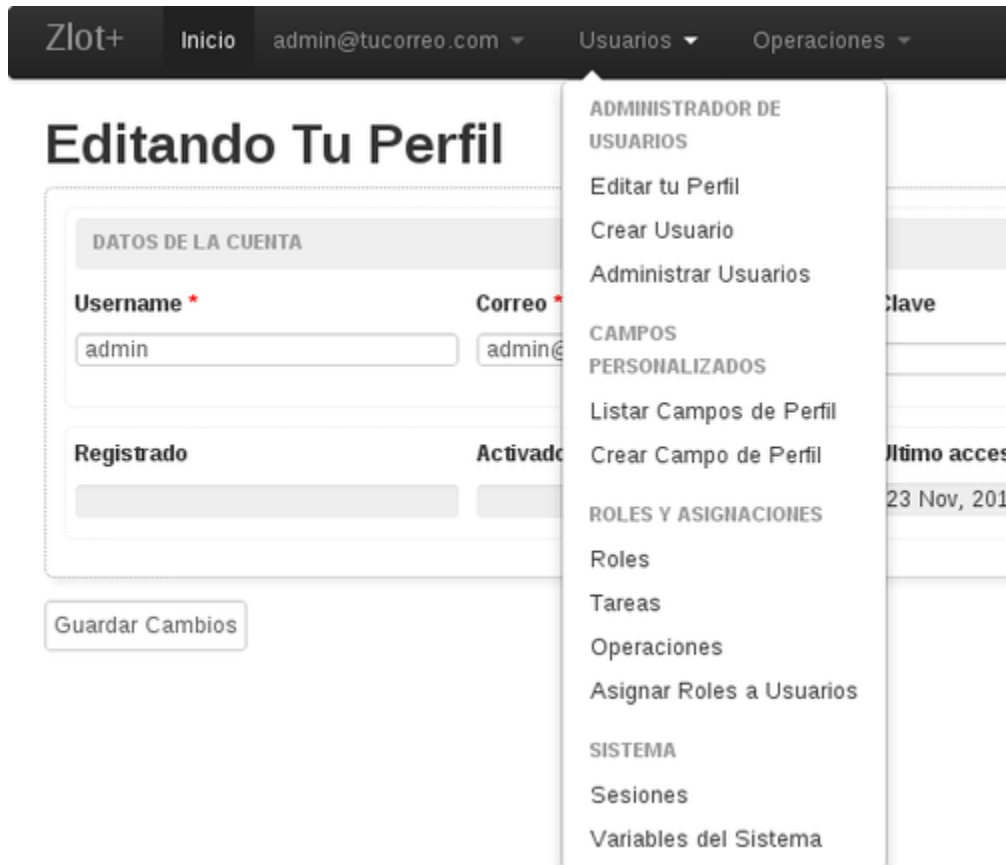


Para lograr esto se usa lo que yo llamo abreviadamente SDDA (Sintaxis De Descripción Avanzada), la cual no es sino una sintaxis que Cruge reconoce en la descripción de una TAREA.

Para conocer los detalles sigue este enlace

CRUGE y BOOTSTRAP

Cruge puede incorporarse perfectamente con bootstrap, no necesitas nada especial para Cruge, puedes ver el siguiente ejemplo de `\\layouts\column1` (<http://pastebin.com/tY65k9UE>) para que conozcas cómo este layout implementa a bootstrap. En este otro ejemplo puedes ver cómo es el componente `uimanager` (<http://pastebin.com/tBdxSDJj>) (que es usado dentro de `\\layouts\column1`) el cual le pasa los menuitems de cruge (y otros mas).



CONECTAR CRUGE CON FACEBOOK Y GOOGLE

Pues bien, tenemos para ello a CrugeConnector. Para saber como conectar ambas partes y lograr esta funcionalidad puedes leer este documento: (Conectar Cruge con CrugeConnector)

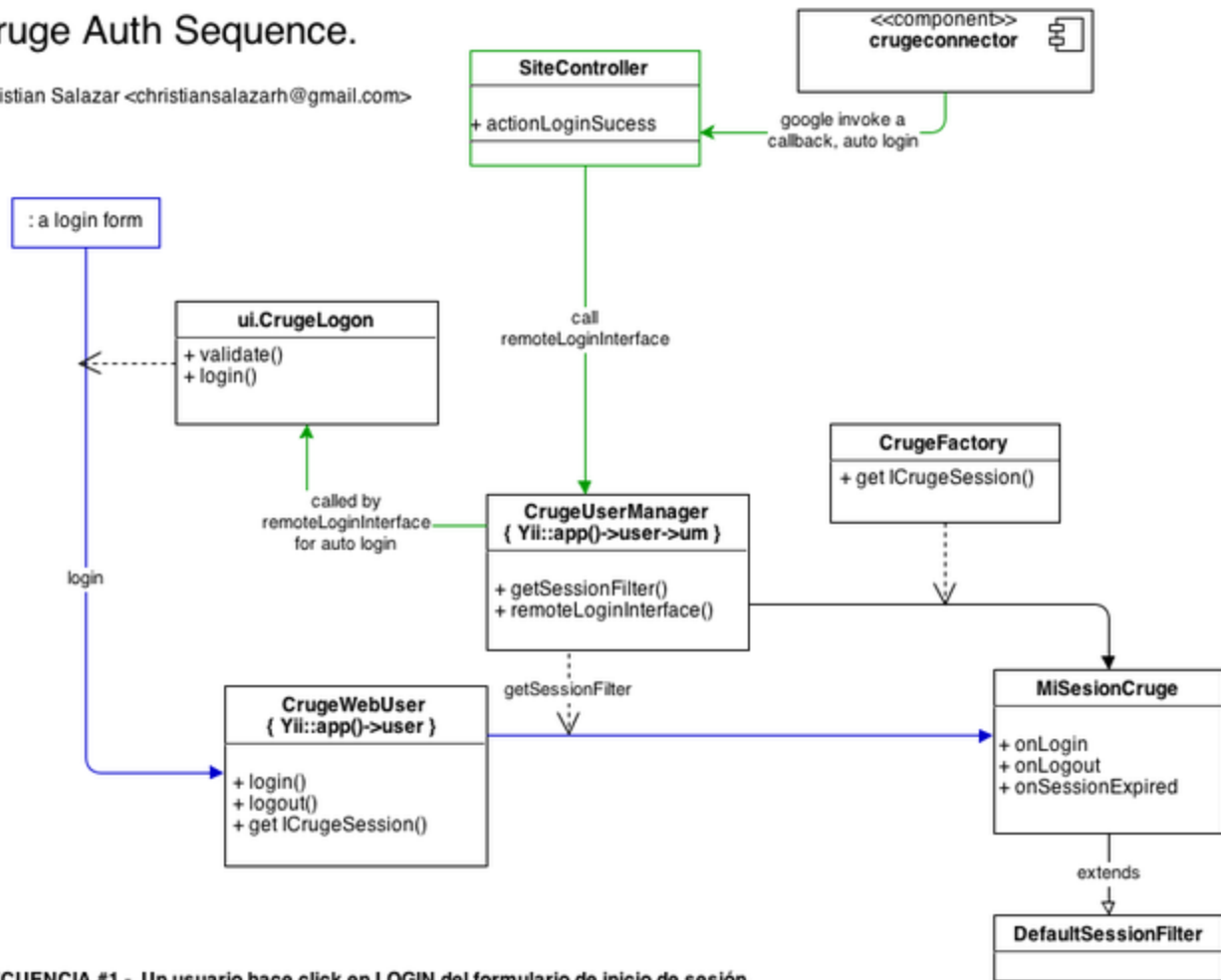
DIAGRAMAS UML DE CRUGE

Es importante conocer cómo está diseñado Cruge, para esto proveeré tres diagramas importantes, no son todos, hay más, pero estos son los indispensables para conocer qué sucede cuando se hace click en "Login":

Diagrama de Clases / Secuencia de Cruge y MiSesionCruge

Cruge Auth Sequence.

Christian Salazar <christiansalazarh@gmail.com>



SECUENCIA #1 - Un usuario hace click en LOGIN del formulario de inicio de sesión.

- 1.1 loginform hace instancia \$model basada en CrugeLogon. para validar credenciales hace model.validate, y tras éxito invoca a model.login().
- 1.2 model.login() invoca a Yii::app()->user->login(), solicita a CrugeUserManager (getSessionFilter) al filtro de sesion activo.
- 1.3 CrugeUserManager, le pide al Factory que le de el filtro de sesion registrado en CONFIG MAIN, por tanto ésta (Factory) le da una instancia de MiSesionCruge.
- 1.4 de regreso a CrugeWebUser.login, se usa el filtro para informar el login exitoso.

SECUENCIA #2 - Login automatico tras una exitosa autentificación remota hecha mediante CrugeConnector usando a un proveedor de autentificación (google, facebook, por ejemplo).

- 2.1 Google ha enviado una llamada de regreso a la aplicacion mediante un callback (google-callback.php, ver CrugeConnector doc.)
- 2.2 CrugeConnector invoca a SiteController::actionLoginSuccess
- 2.3 actionLoginSuccess invoca a CrugeUserManager::remoteLoginInterface (Yii::app()->user->um->remoteLoginInterface) quien mediante otro metodo privado procesa el login "como si hubiese sido hecho mediante el formulario normal", es decir invocando a CrugeLogon::login()
- 2.4 CrugeLogon::login() repite el mismo proceso de la SECUENCIA #1 para llegar a MiSesionCruge.

TIP PARA COMPRENDER EL DIAGRAMA:

Esto se llama "clase de asociación". Indica que: "A se conecta a B mediante el uso de X"



Diagrama de Clases Cruge

En este primer diagrama se muestran las clases que están involucradas en el proceso de autenticación (junto a su relación con otras clases).

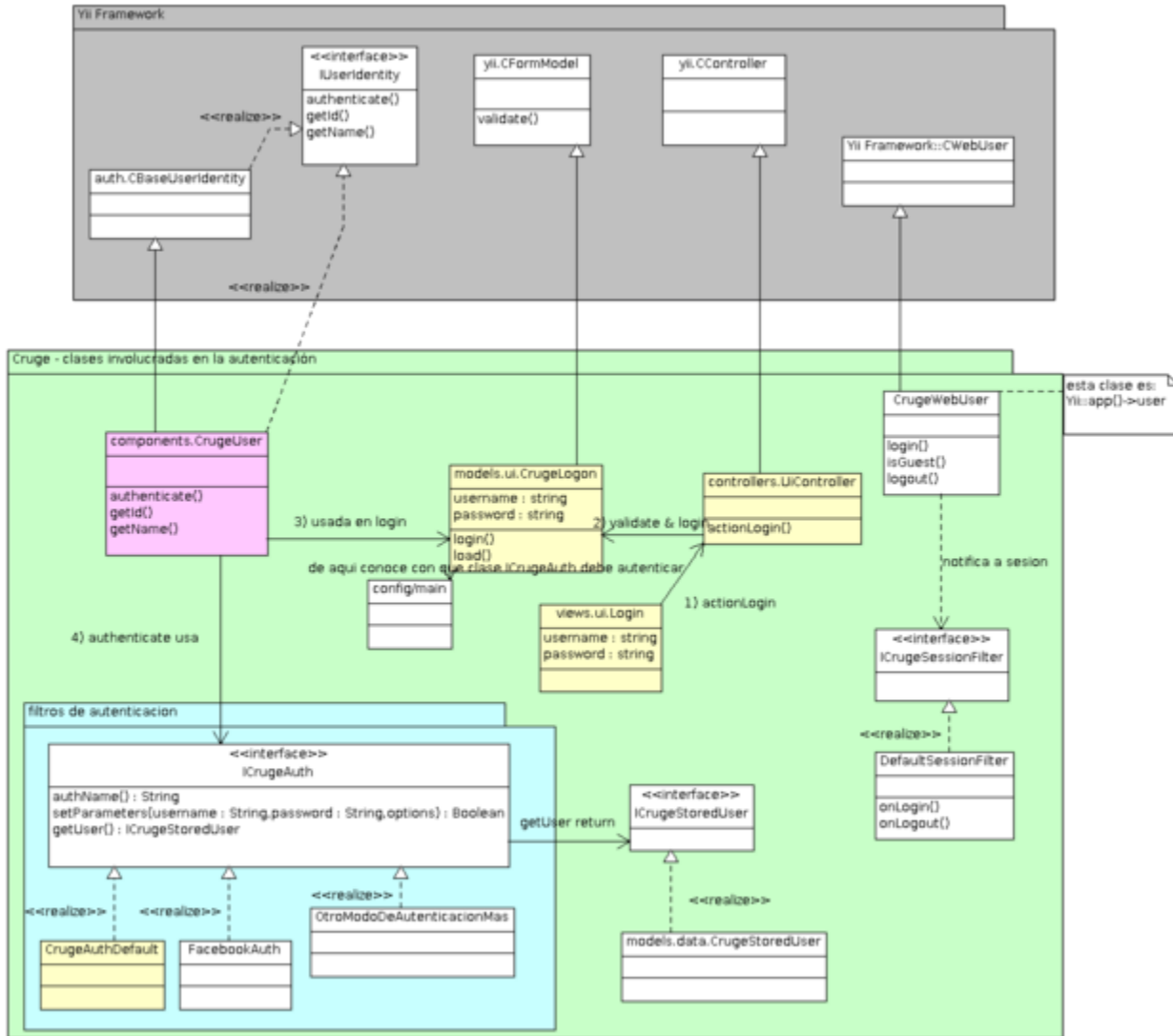


Diagrama Actividad (caso "login")

En este diagrama de actividad puedes observar que sucede en líneas generales cuando se presiona el botón login.

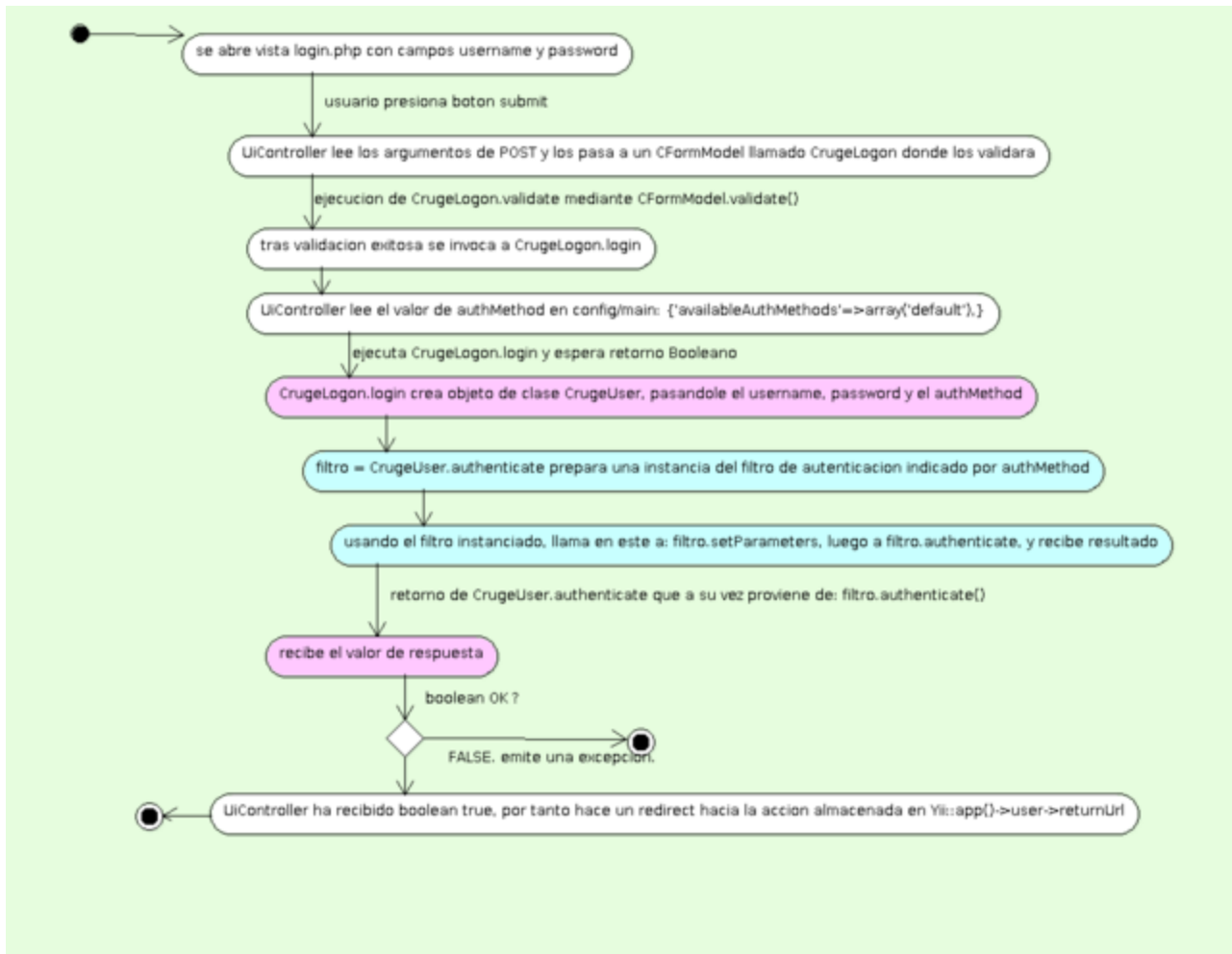


Diagrama de Secuencia (caso "login")

aquí hay una secuencia en el tiempo y una vista de las clases involucradas, este diagrama se lee de izquierda a derecha, arriba en los cuadros grandes al tope se listan las clases involucradas, de cada clase sale una línea vertical larga de la cual a su vez salen flechas hacia otras clases, esas flechas son acciones a realizarse, llamadas, etc.

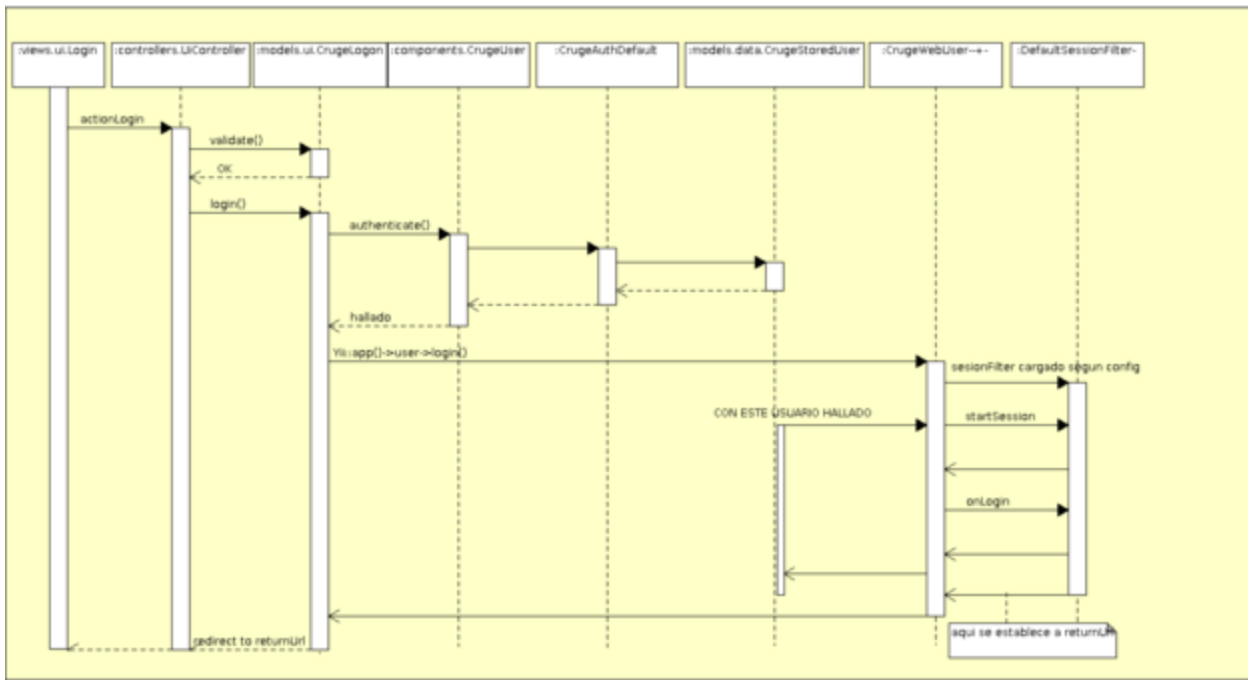
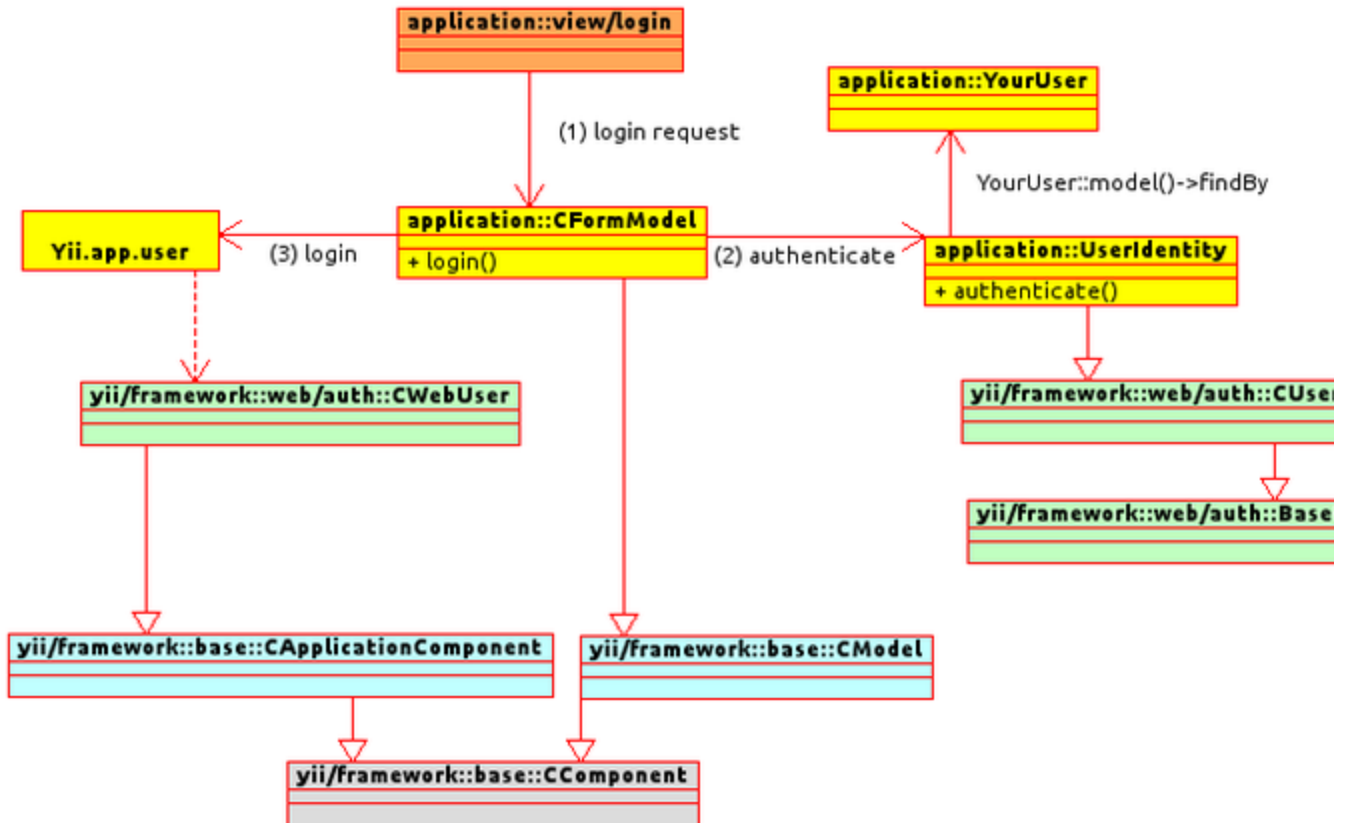


Diagrama de Clases de Yii Auth Base

este es el diagrama de clases de Yii Auth "estandar" (de paquete). Sirve para entender mejor a cruge.

Yii Auth Class Diagram

christiansalazarh@gmail.com - yiiframeworkenespanol.org



Obtenido de "<http://coquito/yiies/wiki/index.php/Cruge>"

- Esta página fue modificada por última vez el 30 ago 2013, a las 22:54.