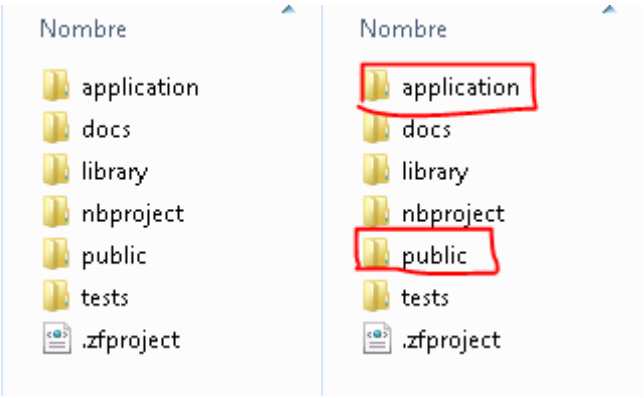
MANUAL DEL PROGRAMADOR

Para la creación de la página web se utilizó Zend Framework y bootstrap. Se utilizó el modelo vista controlador (MVC)

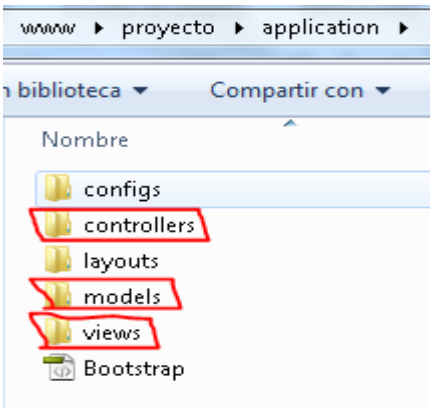
El proyecto cuenta con las siguientes subcarpetas:



Las más importantes son las carpetas aplication y public

Carpeta Aplication

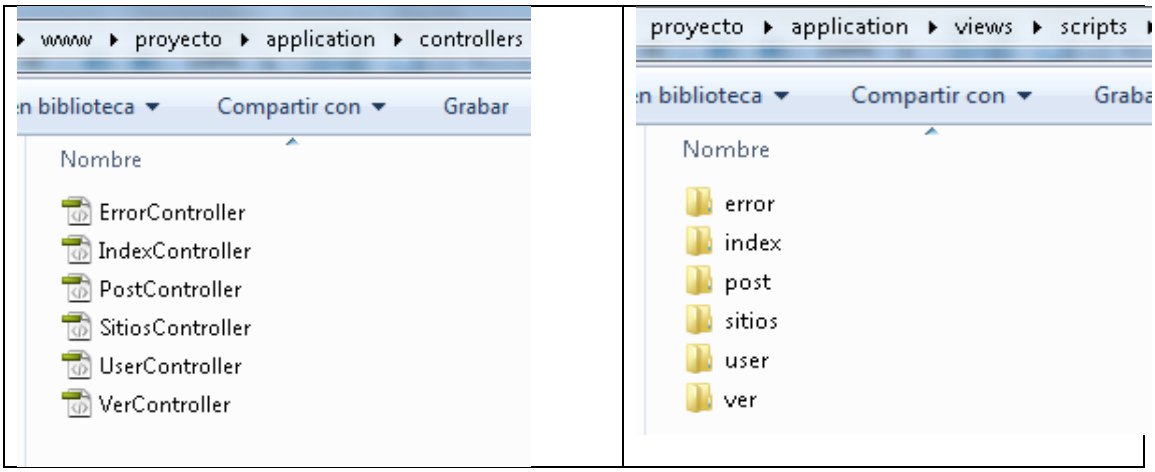
En esta carpeta se encuentran las siguientes subcarpetas.



Como se aprecia en tenemos las carpetas controllers, models y views. Las cuales sirven para trabajar con el modelo MVC.

Cada controlador tiene que tener una vista, y en el modelo es donde se realiza la lógica como las consultas a la base de datos.

En la siguiente figura se puede apreciar como por cada controlador existe una vista



1. Carpeta controllers

En esta carpeta se encuentra todos los controladores de la página.

1.1. ErrorController.php

En este archivo solo contamos con una clase y dos funciones

```php
<?php

class ErrorController extends Zend_Controller_Action
{

    public function errorAction()
    {
        $errors = $this->_getParam('error_handler');

        if (!$errors || !$errors instanceof ArrayObject) {

        switch ($errors->type) {

        // Log exception, if logger available
        if ($log = $this->getLog()) {

        // conditionally display exceptions
        if ($this->getInvokeArg('displayExceptions') == true) {

        $this->view->request   = $errors->request;
    }

    public function getLog()
    {
        $bootstrap = $this->getInvokeArg('bootstrap');
        if (!$bootstrap->hasResource('Log')) {
        $log = $bootstrap->getResource('Log');
        return $log;
    }
```

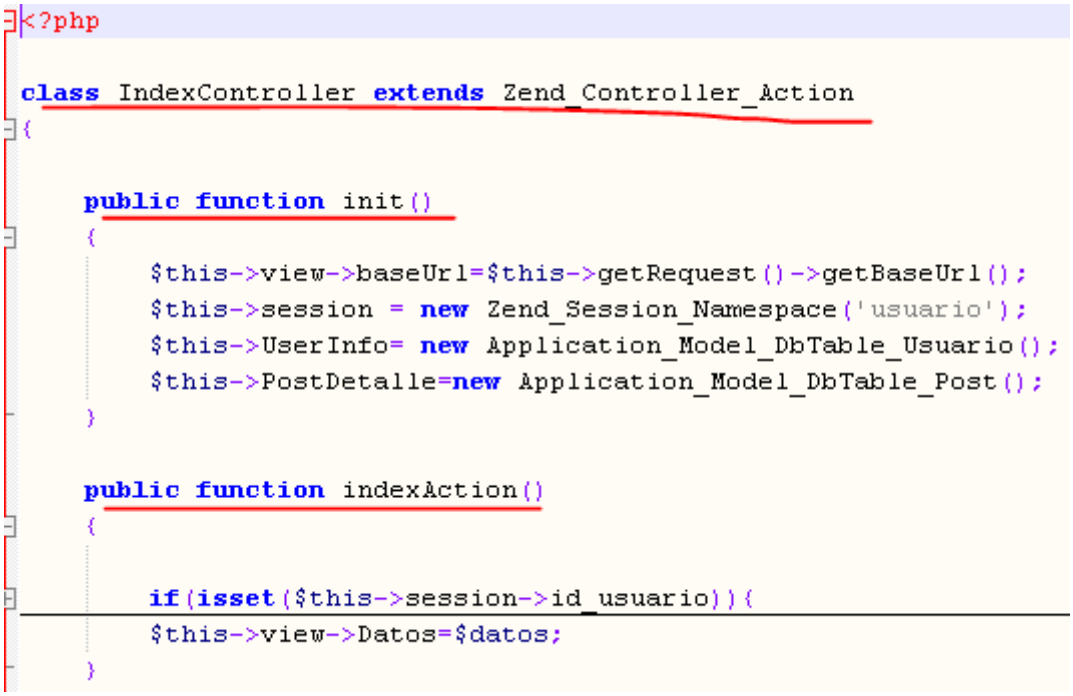
La clase ***class ErrorController extends Zend_Controller_Action***, contiene todos los métodos y funciones que sirven en caso de que haya un error en la ejecución de la pagina

La función **public function errorAction(),** verifica si ocurre alguna falla, si detecta un error lo captura en un array y muestra un mensaje con los errores encontrados.

La función **public function getLog(),** Verifica se realiza con éxito al iniciar la ejecución del sistema.

1.2. IndexController.php

En este archivo contamos con una clase y 2 funciones.



```php
<?php

class IndexController extends Zend_Controller_Action
{

    public function init()
    {
        $this->view->baseUrl=$this->getRequest()->getBaseUrl();
        $this->session = new Zend_Session_Namespace('usuario');
        $this->UserInfo= new Application_Model_DbTable_Usuario();
        $this->PostDetalle=new Application_Model_DbTable_Post();
    }

    public function indexAction()
    {

        if(isset($this->session->id_usuario)){
        $this->view->Datos=$datos;
    }
```

La clase **class IndexController extends Zend_Controller_Action,** es la clase principal de la página, es la que se ejecuta por defecto.

En la función **public function init(),** se encuentran todas las variables necesarias para que la página se ejecute con las características como se programó.

En la función **public function indexAction(),** se encuentra el código para verificar la sesión de un usuario, para poder cargar los datos respectivos a este usuario.

1.3 PostController.php

Como observamos en los dos archivos anteriores nos damos cuenta que en ambos hubieron 2 funciones el **init()** y en **IndexAction** estas 2 funciones siempre estarán en cualquier controlador, ya que por defecto siempre se ejecutara primero esas dos funciones.

```php
<?php

class PostController extends Zend_Controller_Action
{

    public function init()
    {

    public function indexAction()
    {

    public function guardarAction()
    {
if(isset($this->session->id_usuario)){
    }

}
```

Como se puede observar, solo existe una clase **class PostController extends Zend_Controller_Action,** la función **public function guardarAction()**, se encarga de guardar, toda la información de una publicación en la base de datos, siempre y cuando el usuario este logueado, en caso contrario carga la página principal otra vez.

1.4 SitiosController.php

Este archivo contiene solo una clase, al igual que las enteriores, como ya os abras dado cuenta, todos los controladores tienen solo una clase con el mismo nombre del archivo.

```php
<?php

class SitiosController extends Zend_Controller_Action
{

    public function init()
    {

    public function indexAction()
    {

    public function paisAction()
    {

    public function regionAction()
    {

    public function provinciaAction()
    {

    public function distritoAction()
    {
```

Las funciones **public function paisAction(), public function regionAction(), public function provinciaAction() y public function distritoAction()** , capturan información de la base de datos del país, región, provincia y distrito respectivamente. Para poder ser usado cuando se llame a esta función.

1.5 UserController.php

En este archivo encontramos las siguientes funciones

```
<?php

class UserController extends Zend_Controller_Action
{

    public function init()
    {

    public function indexAction()
    {

    public function logueoAction()
    {

    public function registroAction()
    {

    public function logoutAction()
    {

    public function configAction()
    {

    public function subirAction()
    {
```

La función **public function logueoAction(),** es la que se ejecuta cuando un usuario intenta loguearse, verifica si los datos son correctos en la base de datos.

La función **public function registroAction(),** es la que se ejecuta cuando una alguien quiere registrarse, verifica que todo este correcto, y guarda la información en la base de datos, generando un nuevo usuario del sistema.

La función **public function configAction(),** se encarga de las actualizaciones que el usuario realiza a sus datos personales, y las actualiza en la base de datos.

La función **public function subirAction(),** se encarga de subir una foto a la base de datos, de acuerdo al usuario logueado.

1.6 VerController.php

Este archivo simplemente se encarga de la visualización de los datos en la página, trae los datos necesarios de la base de datos y la muestra en la página.

2. Carpeta Models

2.1.Post.php

En este archivo encontramos las funciones que se encargar de conectarse con una tabla de la base de datos, en este caso con la tabla Post, en función **function __construct(),** se crea una instancia a la tabla post. Y en la función **public function guardar(),** es en donde se utiliza la instancia que se hizo para guardar datos en la tabla Post.

```php
<?php

class Application_Model_Post
{
  private $_table;

function __construct(){
    $this->_table=new Application_Model_DbTable_Post();

}

public function guardar($id_post,$titulo,$img,$descripcion,$id_usuario,$permiso,
$categoria)
{
}
```

2.2.Usuario.php

Este archivo tiene una estructura similar al anterior, solo que esta esta instanciada a la tabla usuario.

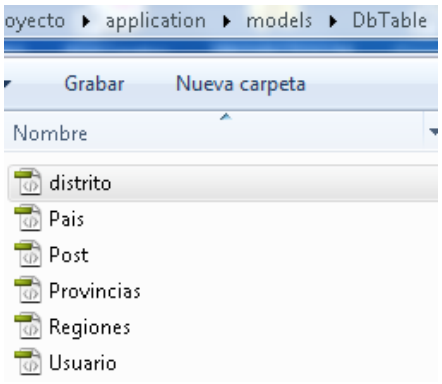
```php
<?php

class Application_Model_Usuario
{

 private $_table;
 function __construct(){
    $this->_table=new Application_Model_DbTable_Usuario();

}
public function guardar($id_usuario,$nombre,$apellido,$password,$rol
,$alias,$pais,$region,$provincia,$distrito)
{
}
```

Dentro de la carpeta models se encuentra una sub carpeta DbTable en donde se encuentran los siguientes archivos:



2.3 carpeta DbTable

2.3.1 distrito.php

```php
<?php

class Application_Model_DbTable_Distrito extends Zend_Db_Table_Abstract
{

    protected $_name = 'distrito';


public function All($Id)
{
    $db=$this->getAdapter();
    $select=$db->select();
    $select->from(array('a'=>$this->_name));
    $select->where('a.id_provincia=?',$Id);
    return $db->fetchAll($select);
}
}
```

En este archivo es en donde nos conectamos a la base de datos y la tabla distrito, también encontramos una función **public function All($Id)**, la cual hace una consulta a la base de datos y trae los datos solicitados.

2.3.2 País.php

Este archivo es similar al anterior, solo que se conecta a la tabla País, de la misma forma realiza una consulta a la base de datos y devuelve los resultados correspondientes.

```php
<?php

class Application_Model_DbTable_Pais extends Zend_Db_Table_Abstract
{

    protected $_name = 'pais';


public function All($Id)
{
    $db=$this->getAdapter();
    $select=$db->select();
    $select->from(array('a'=>$this->_name));
    $select->where('a.id_pais=?',$Id);
    $select->order(array('a.id_post DESC'));
    return $db->fetchAll($select);
}
}
```

2.3.3 Post.php

En este archivo, se hace lo mismo que en los anteriores, pero con la tabla Post. A diferencia de las anteriores en este archivo existen más funciones.

```php
<?php

class Application_Model_DbTable_Post extends Zend_Db_Table_Abstract
{

    protected $_name = 'post';


public function Distrito($distrito)
{
public function Provincia($provincia)
{
public function Region($region)
{
public function Pais($pais)
{
public function Id($Id)
{
public function ultimos10()
{
}
```

La función **public function Distrito($distrito),** realiza una consulta a la base de datos de acuerdo al distrito del usuario.

La función **public function Provincia($provincia),** realiza una consulta a la base de datos de acuerdo a la provincia del usuario.

La función **public function Region($region),** realiza una consulta a la base de datos de acuerdo a la region del usuario.

La función **public function Pais($pais),** realiza una consulta a la base de datos de acuerdo al país del usuario.

La función **public function Id($Id),** Realiza una consulta a la base de datos de acuerdo al id del post o publicación del usuario.

La función **public function ultimos10(),** Realiza una consulta a la base de datos, captura las ultimas 10 filas o publicaciones que realizo el usuario.

2.3.4 Provincias.php

En este archivo es en donde nos conectamos a la base de datos y la tabla distrito, también encontramos una función **public function All($Id)**, la cual hace una consulta a la base de datos y trae los datos solicitados.

```php
<?php

class Application_Model_DbTable_Regiones extends Zend_Db_Table_Abstract
{

    protected $_name = 'regiones';


public function All($Id)
{
    $db=$this->getAdapter();
    $select=$db->select();
    $select->from(array('a'=>$this->_name));
    $select->where('a.id_pais=?',$Id);
    return $db->fetchAll($select);
}
}
```

2.3.5 Regiones.php

Este archivo hace lo mismo que el anterior solo que con la tabla regiones.

2.3.6 Usuario.php

En este archivo se encuentra todas las funciones de consulta a la base de datos respecto al usuario.

public function User_Dato($id_user), selecciona los datos del usuario de acuerdo a su id usuario.

public function User_Id($alias), selecciona los datos del usuario de acuerdo a su sobrenombre o alias.

public function Todo(), selecciona todos los datos del usuario

```php
<?php
class Application_Model_DbTable_Usuario extends Zend_Db_Table_Abstract
{
    public $_name = 'usuario';

public function User_Dato($id_user){
public function User_Id($alias){
public function Todo(){
public function Existe_alias($alias){
}
```

AHORA MOSTRAREMOS EL MANUAL DEL PROGRAMADOR DE LA APLICACIÓN MOVIL

public class
LoginActivity
extends **Activity**
android.app.Activity;

Descripcion General
Clase que muestra la pantalla de ingreso al sistema, gestiona los elementos que se muestran en pantalla. Hace una consulta al servidor para verificar si los datos ingresados por el usuario con correctos, ésta consulta se hace a través de una clase Asíncrona denominada Conectar.

Resumen	
Métodos Protegidos	
	onCreate(Bundle savedInstanceState) Metodo llamado por el Sistema Operativo Android, inicializa todas las variables de la interfaz.
	onActivityResult(int requestCode, int resultCode, Intent data) Método llamado por el Sistema Operativo Android cuando otra interfaz devuelve información para ser usada por esta interfaz.
Metodos Privados	
	iniciar() Método instancia los objetos de la vista en variables del sistema.
	verificarServer(String unAlias,String unPassword) Verifica si los datos ingresados pro el usuario son correctos haciendo una consulta en el servidor.
	CargarPantallaPrincipal() Abre la pantalla principal **MainActivity**

static boolean	**existe**(String[] archivos, String archbusca) Comprueba si existe un determinado archivo en la memoria.
	grabar() Graba datos en un fichero de texto plano.

Clases Privadas	
	Conectar extends AsyncTask<String, Void, String> Clase que realiza una consulta al servidor en segundo plano, para no afectar la ejecución de la aplicación, además para que sea compatible con nuevas versiones de android, ya que de no usarla provocará demora en los procesos y el Sistema Operativo Android podría interrumpir su ejecución ocasionando el cierre de la aplicación.

Métodos Protegidos
protected void **onCreate**(Bundle savedInstanceState) Permite que la interfaz se muestre en pantalla, además de cargar todos los elementos visibles.

Parametros	
savedInstanceState	La información de los elementos que serán usados en la vista, es proporcionada por el Sistema Operativo Android.

protected void **onActivityResult**(int requestCode, int resultCode, Intent data) Analiza Los datos devueltos y finaliz
la Ejecucion de la apliccion.

	Parametros	
	requestCode	Codigo de la clase que envía los resultados
	resultCode	Codigo que informa la normal si la clase que envía los resultados realizo sus procesos correctamente.
	data	Datos enviados por la clase que envia los resultados.

Metodos Privados
private void **iniciar**() Instancia los elementos de la Interfaz en variables del sistema para que puedan ser usados posteriormente.
private void **verificarServer**(String unAlias,String unPassword) Crea una instancia de la clase Asíncrona **Conectar** para realizar una consulta al servidor.

Parámetros	

unAlias	Alias que identifica al usuario
unPassword	Contraseña del usuario

Variables Privadas	
String alias	Alias que representa al usuario
String password	Contraseña del usuario
String nombre	Nombre del Usuario
String apellido	Apellido del usuario
Stirng rol	Rol que cumple el usuario
EditText txtEditAlias	Campo de texto que recibirá el Alias del usuario desde la interfaz
EditTExt txtEditPass	Campo de texto que recibirá la Contraseña del usuario desde la interfaz
Button btnIniciarSesion	Botón visible en la interfaz para iniciar la Sesión en la aplicación

private void **CargarPantallaPrincipal**()
Carga y muestra la pantalla Principal, la que servirá para subir las imágenes al servidor

private boolean **existe**(String[] archivos, String archbusca)
Comprueba la existencia de un archivo de texto plano en la memoria del celular

Parametros	
archivos	Lista de archivos presentes en la memoria
archbusca	Nombre del archivo que se desea encontrar
Returns	
Regresa un valor booleano **true** si se encuentra el fichero buscado caso contrario retorna **false**	

private void **grabar**()
Graba los datos importantes para la aplicación en un fichero de texto plano.

Clases Privadas
private class
conectar
extends **AsyncTask<String, Void,String>**
android.os.AsyncTask;

Descripcion General

Clase que realiza una consulta en la base de datos en un proceso de segundo plano, si los procesos se ejecutan co
normalidad hará un llamado a la ventana principal.

Resumen	
Constructores	
	Conectar(Activity activity) Constructor de la clase asíncrona recibe la instancia de la clase que la llama.
Metodos Protegidos	
String	**doInBackground**(String... urls) Recibe un array de direcciones (URLs) para realizar la consulta a la base de datos, devuelve un String como resultado de la consulta.
	onPostExecute(String feed) Realiza cambios desacuerdo a los resultados retornados por **doinBackground**

Constructores

public **Conectar**(Activity activity)
Construye un objeto Conectar , recibir la instancia de la actividad que lo llama.

Metodos Protegidos

protected String **doInBackground**(String... urls)
Método que se ejecuta en segundo plano, realiza una consulta a la base de datos para contrastar la información registrada por el usuario y devuelve una cadena String con el resultado proporcionado por el Servidor Web La llamada a este método está controlado por el sistema Operativo

Parametros	
urls	Array con las direcciones que se usaran para la consulta a la base de datos
Returns	
String	Cadena con el resultado devuelto por el servidor que puede tener un objeto JSON.

protected void **onPostExecute**(String feed)
Método que se encarga de realizar cambios en la aplicación desacuerdo a los datos devueltos por **doinBackground.** La ejecución de este método es controlado por el Sistema operativo Android y siempre se ejecutara después de **doinBackground**

Parametros	
feed	Cadena de resultado enviada por **doinBackground**

Variables Privadas	
private Activity activity	Actividad que representará la clase que hace uso de la Clase asíncrona

public class
MainActivity
extends **Activity**
android.app.Activity;

Descripción General
Clase principal que permite al usuario subir imágenes y datos al servidor web, crea en pantalla un formulario mostrando las opciones con las que el usuario cuenta y los campos que este deberá llenar

Resumen	
Metodos Públicos	
int	**uploadFile**(String sourceFileUri) Sube una imagen desde la galería al servidor y devuelve un entero corespondiente al proceso
static boolean	**onCreateOptionsMenu**(Menu menu) Metodo que carga el menu para la interfaz
static boolean	**onOptionsItemSelected**(MenuItem item) Evento que corresponde a presionar en un iten del menu.
String	**getPath**(Uri uri) Devuelve una cadena correspondiente a la ubicación de un archivo.
	onBackPressed() Evento que se ejecuta cuando el usuario presiona la tecla **Atrás** del celular

Metodos Protegidos	
	onCreate(Bundle savedInstanceState) Carga e instancia los elementos de la interfaz de usuario
	onActivityResult(int requestCode, int resultCode, Intent data) Se ejecuta cuando una actividad devuelve de datos a esta clase.

	onSaveInstanceState (Bundle outState) Guarda informacion actual
	onRestoreInstanceState(Bundle recEstado) Restablece los valores guardados por **onSaveInstanceState**

Metodos Privados	
	iniciar() Instancia los elementos de la pantalla en variables locales
	defineUpLoadServerUri(String unTitulo,String unaCategoria,String unaDescripcion) Define los datos a enviar al servidor

Clases Privadas	
	UploaderFoto Sube una foto al servidor en segundo plano

Metodos Público
public int **uploadFile**(String sourceFileUri) Sube una imagen seleccionada desde la galería al servidor.

Parametros	
sourceFileUri	Cadena que contiene la dirección (URL) para subir la imagen
Returns	
int	Numero de respuesta dado por el servidor

public boolean **onCreateOptionsMenu**(Menu menu) Carga el menú de la aplicación.

Parametros	
menu	Información de la configuración del menú proporcionado por el sistema operativo.
Returns	
static boolean	**True** si el menu es creado, **false** de ocurrir lo contrario

boolean **onOptionsItemSelected**(MenuItem item)
Evento que se ejecuta cuando el usuario presiona un item del menú.

Parametros	
item	Información del ítem que fue presionado por el usuario
Returns	
static boolean	**True** si se consume el evento, **false** si ocurre lo contrario

public String **getPath**(Uri uri)
devuelve la ubicación de un archivo alojado en la memoria del celular.

Parametros	
uri	Nombre del archivo
Return	
String	ruta del archivo

public void **onBackPressed**()
evento que se ejecuta cada vez que el usuario presiona la tecla **atrás** del celular

Métodos Protegidos
protected void **onCreate**(Bundle savedInstanceState) Permite que la interfaz se muestre en pantalla, además de cargar todos los elementos visibles

Parametros	
savedInstanceState	La información de los elementos que serán usados en la vista, es proporcionada por el Sistema Operativo Android.

protected void **onActivityResult**(int requestCode, int resultCode, Intent data)
Analiza Los datos devueltos y finaliza la Ejecucion de la apliccion.

Parametros	
requestCode	Codigo de la clase que envia los resultados

resultCode	Codigo que informa la normal si la clase que envia los resultados realizo sus procesos correctamente
data	Datos enviados por la clase que envia los resultados.

Metodos Privados
private void **iniciar**() Instancia los elementos de la Interfaz en variables del sistema para que puedan ser usados posteriormente.
private void **defineUpLoadServerUri**(String unTitulo,String unaCategoria,String unaDescripcion) Crea una instancia de la clase Asíncrona **Conectar** para realizar una consulta al servidor.

Parámetros	
unTitulo	Titulo para la publicación
unaCategoria	Categoría de la publicación
unaDescripcion	Descripción de la publicación

Variables Privadas	
String alias	Alias que representa al usuario
String password	Contraseña del usuario
String nombre	Nombre del Usuario
String apellido	Apellido del usuario
Stirng rol	Rol que cumple el usuario
TextView messageText	Campo de texto que recibirá el Alias del usuario desde la interfaz
Calendar c	Campo de texto que recibirá la Contraseña del usuario desde la interfaz
Button btnselctpic	Boton para seleccionar imagen de la galería
Button btnViewCam	Boton para abrir la cámara y tomar una foto
Button uploadButton	Botón para subir los datos al servidor
EditText txtEditTitulo	Campo para poner el título de la publicación
EditText txtEditCategoria	Campo para poner la categoría de la publicación
EditText txtEditDescripcion	Campo para poner la descripción de la publicación
ImageView imageview	Componente de la vista en la que se muestra el logo de la aplicación
int serverResponseCode	Entero para almacenar la respuesta del servidor
ProgressDialog dialog	Mensaje que se muestra en pantalla para informar sobre el proceso de subida
String uploadServerIUri	Url para el servidor en formato Uri
String urlBase	Dirección(URL) base para la consulta al servidor
string imagePath	Ruta de la imagen alojada en la galería del celular
string imagePathCam	Ruta de ubicación de la foto tomada por la cámara del celular

Clases Privadas
private class
UploaderFot
extends **AsyncTask<String, Void, Void>**

android.os.AsyncTask;

Descripcion General
Clase que sube la imagen al servidor en un proceso de segundo plano.

Resumen	
Constructores	
	Conectar(String uploadServerUri) Constructor de la clase asíncrona recibe la url que se usara para subir la imagen al servidor.
Metodos Protegidos	
	doInBackground(String... urls) Recibe un array de strings (URLs) el primero valor contiene el nombre de la foto que se subirá, este método corre en segundo plano subiendo la foto al servidor.
	onPreExecute(String feed) Carga las variables que serán usadas por la clase
	onPostExecute(Void result) Elimina el mensaje que se ve en pantalla.

Constructores
public **UploaderFoto**(String uploadServerUri) Construye un objeto UploaderFoto, recibir la url para subir la foto al servidor.

Metodos Protegidos
protected Void **doInBackground**(String... urls) Método que se ejecuta en segundo plano, sube la información proporcionada por el usuario y sube la imagen al Servidor Web, la llamada a este método está controlada por el sistema Operativo.

Parametros	
urls	Array con las direcciones que se usaran para la consulta a la base de datos
Returns	
String	Cadena con el resultado devuelto por el servidor que puede tener un objeto JSON

protected void **onPreExecute**() Crea un mensaje para informar el estado del proceso de subida de la imagen

protected void **onPostExecute**(Void result)
elimina el mensaje que se muestra en pantalla, su ejecución es controlada por el sistema operativo y se ejecuta siempre después de **doinBackground**

Variables Privadas	
ProgressDialog pDialog	Variable para controlar el mensaje de progreso del proceso
String miFoto	Nombre de la foto
String uploadServerUrl	Variable que contendrá la dirección(URL) para la conexión con le servidor