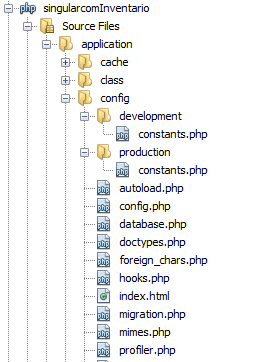
**ESTANDARES DESARROLLO SINGULARCOM**

**SINGES**

VERSIONAMIENTO GIT

|  |  |
| --- | --- |
| **v17.07.27.1** | |
| **v** | Versión |
| **17** | Año |
| **07** | Mes(Importante el 0 para conservar el orden cronológico) |
| **27** | Día |
| **01** | Cambio del día (En caso de múltiples cambios el mismo día) |

ENTORNOS DE DESARROLLO



* Se independizar las constantes que se van a usar dependiendo el entorno a usar
  + development: Entorno de pruebas, conexiones a DB (Bootes), restapi test, etc.
  + Production: Entorno de producción, conexión a DB (castor), restapi, etc.

En archivo **index.php** de codeigniter (raíz de cada proyecto, se indica el entorno a utilizar)

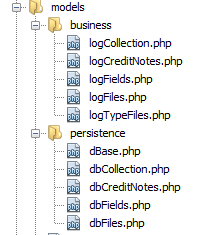
MODELOS

BUSINESS (Modelos de negocio)

* Estos modelos en su contenido deben tener toda la lógica del proceso
* Estos modelos solo deben estar instanciados por los controladores
* Prefijo log

PERSISTENCE (Modelos de persistencia)

* Estos modelos son exclusivamente los que se encargan de conexión a la DB, (consultas, ingresos, etc.)
* Estos modelos solo deben estar instanciados por los modelos de negocio (business)
* Prefijo db

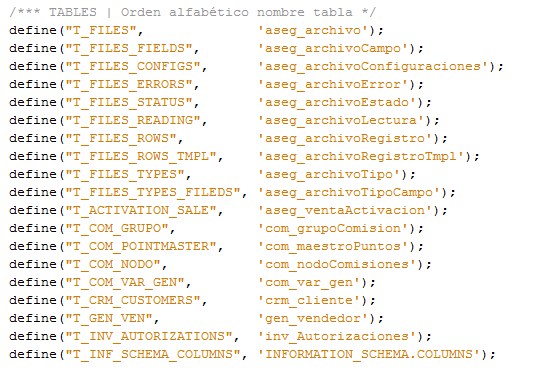


NOTA

* Los controladores no deben tener lógica.
* Los controladores NO deben tener instanciado el modelo de persistencia

NOMBRE DE TABLAS DB

* Nombramiento de tablas (Orden alfabético)

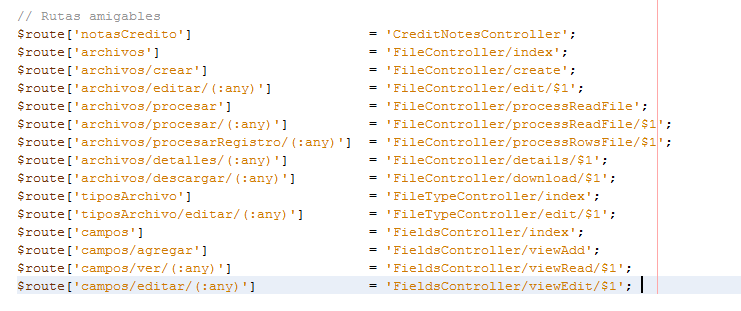


A nivel de modelo:



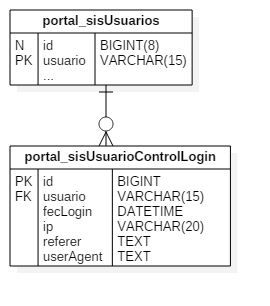
RUTAS AMIGABLES

Archivo ubicado en el siguiente directorio: *Aplication/config/routes.php*



* Ordenamiento por Nombre de controlador en orden alfabético

**ESTANDARIZACIÓN DB**



* Nombramiento en español
* Nombre de tablas con prefijo a la funcionalidad (cf\_ aseg\_) seguido con el nombre de la tabla en camelCase Ejemplo: **aseg\_archivos** (En plural)
* Nombramiento de attributos en camelCase (en singular) Ejemplo: **fechaIngreso**
* Attributos de llaves foráneas en camelCase (en singular) Ejemplo: **idArchivo**

**IMPLEMENTACIÓN TRANSACCIONES**

Las transacciones son un conjunto de procesos que deseamos llevar a cabo, para comprobar que todo se lleve a cabo podremos hacer lo siguiente en codeigniter.

public function mi\_transaccion() {

//empezamos una transacción  
 $this->db->trans\_begin();

$this->db->query('select \* from usuarios');  
 $this->db->query('insert into usuario');  
 $this->db->query('update usuarios');

//comprobamos si se han llevado a cabo correctamente todas las consultas  
 if ($this->db->trans\_status() === FALSE) {

//si ha habido algún error lo debemos mostrar aquí  
 $this->db->trans\_rollback();

}else{

//en otro caso todo ha ido bien  
 $this->db->trans\_commit();  
 }

}

**CONSULTAS DB**

* En el modelo de persistencia las consultas se realizan a través de el driver de conexión del framework, en la medida de lo posible realizar la consulta a través del contructor de consultas, no dejar código quemado.

|  |  |  |
| --- | --- | --- |
| **SELECT** | **->**select('campo1, campo2') | *SELECT campo1, campo2* |
| **SELECT\_MAX** | **->**select\_max('age') | *SELECT MAX(age) as age* |
| **SELECT\_MIN** | **->**select\_min('age') | *SELECT MIN(age) as age* |
| **SELECT\_AVG** | **->**select\_avg('age') | *SELECT AVG(age) as age* |
| **SELECT\_SUM** | **->**select\_sum('age') | *SELECT SUM(age) as age* |
| **FROM** | **->**from('table') | *SELECT \* FROM table* |
| **JOIN** | **->**join('comments', 'comments.id = blogs.id') | *SELECT \* FROM blogs JOIN comments ON comments.id = blogs.id* |
| **WHERE** | **->**where('name', $name) | *WHERE name = 'Joe'* |
| **OR\_WHERE** | $this**->**db**->**where('name !=', $name);  $this**->**db**->**or\_where('id >', $id); | *WHERE name != 'Joe' OR id > 50* |
| **WHERE\_IN** | $names **=** **array**('Frank', 'Todd', 'James');  $this**->**db**->**where\_in('username', $names) | *WHERE username IN ('Frank', 'Todd', 'James')* |
| **OR\_WHERE\_IN** | $names **=** **array**('Frank', 'Todd', 'James');  $this**->**db**->**or\_where\_in('username', $names); | *OR username IN ('Frank', 'Todd', 'James')* |
| **WHERE\_NOT\_IN** | $names **=** **array**('Frank', 'Todd', 'James');  $this**->**db**->**where\_not\_in('username', $names); | *WHERE username NOT IN ('Frank', 'Todd', 'James')* |
| **OR\_WHERE\_NOT\_IN** | $names **=** **array**('Frank', 'Todd', 'James');  $this**->**db**->**or\_where\_not\_in('username', $names); | *OR username NOT IN ('Frank', 'Todd', 'James')* |
| **LIKE** | **->**like('title', 'match', 'both');  **// before|after|both posición %** | *WHERE `title` LIKE '%match%' ESCAPE '!'* |
| **LIKE** | $array **=** **array**('title' **=>** $match, 'page1' **=>** $match, 'page2' **=>** $match);  $this**->**db**->**like($array); | *WHERE `title` LIKE '%match%' ESCAPE '!' AND `page1` LIKE '%match%' ESCAPE '!' AND `page2` LIKE '%match%' ESCAPE '!'* |
| **OR\_LIKE** | $this**->**db**->**like('title', 'match'); $this**->**db**->**or\_like('body', $match); | *WHERE `title` LIKE '%match%' ESCAPE '!' OR `body` LIKE '%match%' ESCAPE '!'* |
| **NOT\_LIKE** | **->**not\_like('title', 'match'); | *WHERE `title` NOT LIKE '%match% ESCAPE '!'* |
| **OR\_NOT\_LIKE** | $this**->**db**->**like('title', 'match');  $this**->**db**->**or\_not\_like('body', 'match'); | *WHERE `title` LIKE '%match% OR `body` NOT LIKE '%match%' ESCAPE '!'* |
| **GROUP BY** | **->**group\_by("title")  **->**group\_by(**array**("title", "date")) | *GROUP BY title*  *GROUP BY title, date* |
| **DISTINCT** | $this**->**db**->**distinct();  $this**->**db**->**get('table'); | *SELECT DISTINCT \* FROM table* |
| **HAVING** | $this**->**db**->**having('user\_id', 45); | *HAVING user\_id = 45* |
| **ORDER\_BY** | **->**order\_by('title', 'DESC'); | *ORDER BY `title` DESC* |
| **LIMIT** | **->**limit(10)  **->**limit(10, 20) | *LIMIT 10*  *LIMIT 20, 10* |
| **GROUP\_START**  **GROUP\_END** | $this**->**db**->**select('\*')**->**from('my\_table')  **->**group\_start()  **->**where('a', 'a')  **->**or\_group\_start()  **->**where('b', 'b')  **->**where('c', 'c')  **->**group\_end()  **->**group\_end()  **->**where('d', 'd')  **->**get(); | *SELECT \* FROM (`my\_table`) WHERE ( `a` = 'a' OR ( `b` = 'b' AND `c` = 'c' ) ) AND `d` = 'd'* |

<https://www.codeigniter.com/user_guide/database/query_builder.html>

**EXCEPCIONES CONSULTAS DB**



* Incluir en las consultas generación de excepción en caso de error, incluyendo:
  + \_\_CLASS\_\_ : Clase en donde se genera el error.
  + \_\_LINE\_\_: Línea en la que se genera la excepción

**ESTANDARES PHP**

VARIABLES

En general el nombramiento de variables, constantes, funciones, etc…, va a ser en Ingles

|  |  |
| --- | --- |
| **VARIABLES**  **$nameReference = 'xxx’;**  **$\_nameReference = 'xxx’;**  $a = ‘x’; $b = ‘x’; $a1 = ‘x’;  $NOMBRE = ‘X’;  $NombEmp = ‘x’; | * Texto descriptivo * propiedades y parámetros **PÚBLICOS** en camelCase (minúscula a excepción de inicio de cada palabra a partir de la segunda) * attributos **PRIVADOS** en camelCase precedidos por un guión bajo |
| **CONSTANTES**  **$**CONN\_DATABASE **= 'bootes’;**  $CONDB = ‘xx’;  $con\_db = ‘X’; | * Texto descriptivo * Las constantes siempre en ALL\_CAPS. Ejemplo: REST\_API\_CONN |
| **FUNCIONES**  public function exportTable($table) {...}  private function \_exportTable($table) {...} | * Texto descriptivo. * Los nombres de los métodos **PÚBLICOS** en camelCase. Ejemplo: nombreDelMetodo() * Los nombres de los métodos **PRIVADOS** en camelCase precedidos por un guión bajo (underscore-prefixed). Ejemplo: \_nombreDelMetodo() |
| **CLASES**  class MySqlBuilder extends ... | * Texto descriptivo. * Los nombres de las clases en MixedCase. Ejemplo: ElNombreDeMiClase |

DOCUMENTACIÓN

Función

/\*\*  
 \* Explicación breve de la función  
 \* Explicación detallada de la función   
 \* **@author** Camilo Quijano <desarrollo2@singularcom.com.co>   
 \* **@date** 24/11/2017  
 \* **@update** 25/11/2017 Camilo Quijano – Ajuste en xyz   
 \* **@method** GET|POST  
 \* **@see** http://example.com/my/bar Relacionar una fuente o contenido   
 \* **@param** int $nameVar1 explicación breve de variable   
 \* **@param** string $nameVar2 explicación breve de variable   
 \* **@throws** Exception En caso de error en consulta (En caso de excepciones)  
 \* **@return** array descripción breve variable de retorno   
 \* **@version** v17.11.24  
 \*/  
public function getItemsRelDocument($nameVar1, $nameVar2) { …

---------------------------------------------------------------------------------------------------

Variables

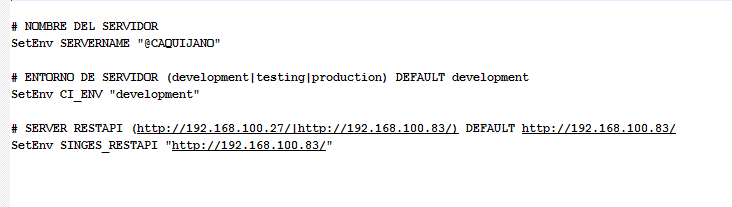
/\*\*  
 \* **@var** string $name Descripción uso de varible  
 \* **@var** string $description Descripción uso de varible   
 \*/  
 protected $name, $description;

PHP Documentor   
<http://docs.phpdoc.org/references/phpdoc/index.html>

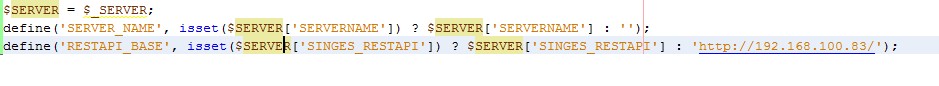
**PROXIMA IMPLEMENTACIÓN**

El nombramiento de Constantes de cada uno de los proyectos se van a definir a partir de una variable de servidor

.**htaccess** en raíz de server



y en cada uno de los proyectos, en la definición de constantes evaluamos inicialmente esta configuración del servidor, en caso contrario tomaría el valor por default



Inicialmente se plantea para:

* Conexión DB
* Urls restapi y/o servidores externos
* Entorno de desarrollo
* Nombre del servidor

**BUENAS PRÁCTICAS**

|  |  |
| --- | --- |
| Al iterar arrays, fija el valor máximo fuera del bucle Cada llamada al método count() aumenta hasta en un 50% el tiempo de ejecución, según el tamaño del array. | **// Mala práctica**  for ($i = 0, $i < count($miArray); $i++){ ... }  **// Buena práctica**  $limite = count($miArray);  for ($i = 0; $i < $limite; $i++){ ... } |
| **Uso ISSET**: Si queremos comprobar que una variable tiene más de una determinada longitud, podemos hacerlo de un modo más efectivo de la siguiente manera: | if (isset($username{5})){  ....  } |

PENDIENTES

* Implementación de excepciones uso general
* Implementación de libraría de mensajes de error

FUENTE:   
<http://docs.phpdoc.org/references/phpdoc/index.html>  
<http://formaciononline.eu/buenas-practicas-para-programar-en-php/>  
<http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_tags.ignore.pkg.html>