Erklärung MVC, Methodenaufruf und php Struktur

# Ordnerstruktur von KiCa

**/php**: ist der Ausgangsordner also die „root“. Beinhaltet unteranderem die **erste index.php**

**/php/\_tutorial**: beinhaltet die Anleitung zum Framework. **Wird gelöscht**!

**/php/application**: beinhaltet alle Dateien und Verzeichnisse die auf **Serverseite** ausgeführt werden

**/php/application/config**: enthält die **config.php**

**/php/application/controller:** beinhaltet alle Klassen spezifischen Controller

**/php/application/libs:** enthält den “**root\_Controller**” und die **application.php**

**/php/application/models:** enthält die Models je Controller ein Model

**/php/views:** enthält pro Seite/Klasse einen Ordner mit verschiedenen php-Dateien für

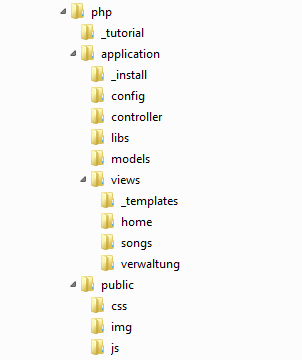
einzelne Seiten

**/php/views/\_templates:** enthält Dateien, die in jedem View aufgerufen werden, wie

Header, Footer etc.

**/php/views/songs:** ist ein Beispielordner des Frameworks, **wird gelöscht**!

**/php/public/:** enthält Dateien die auf Clientseite ausgeführt werden, wie Bilder, Stylesheets und Javascripts



# KiCa MVC

Prinzipiell sieht eine MVC Struktur so aus wie der Name es bereits sagt.

Model

Controller

View

Die Kommunikation erfolgt immer über den Controller, der die Zentrale Schnittstelle darstellt.

Die auszuführenden Methoden liegen im entsprechenden Model. Der Controller ruft in einer Methode die entsprechende Methode im Model auf und übergibt die Variablen. Im View werden die Methoden des Controllers über html Methodenaufrufe realisiert.

**KiCa MVC**

Das hinterlegte Framework umfasst deutlich mehr. Als erstes wird immer die erste index.php – Datei aufgerufen!

/application/config/config.php

/application/libs/controller.php

/application/libs/application.php

erzeugt neues Objekt

index.php

Die **index.php** ruft die **config.php** und den „root“ **controller.php** auf. Ebenso wird ein Objekt von **application.php** erzeugt.

Die **config.php** beinhaltet Einstellungen und Daten zur DB, wie Verbindung, Name und Passwort. Desweiteren wird eine variable *URL* definiert, die Root-URL [http://kica.tld/] beinhaltet. Dabei handelt es sich um eine globale Variable, die von **application.php** aufgerufen wird.

Die **application.php** ist eigentlich ein Router. Hier wird die Url in der Adressleiste auseinander genommen und umgeleitet. Hierdurch ist es möglich Klassen und Mehtoden in der Url abzubilden, anstatt der Ordnerstruktur. Programmiert ist, dass zuerst die root URL [http://kica.tld/] kommt. Danach kommt die Klasse, anschließend folgt die Methode und danach 3 mögliche Parameter.

Beispiel:

http://kica.tld/[Klasse]/[Methode]/[Parameter 1]/[Parameter 2]/...

Es ist möglich mehr Parameter zu konfigurieren, wird wahrscheinlich nicht benötigt.

Der „root“ **controller.php** vererbt an alle Controller im Ordner /application/controller/.

Jeder einzelne Controller lädt benötigte Models und Views direkt.

# Methodenaufruf

**View**

Beginnend bei der **View** Datei: In einer <form> wird eine „action“ hinterlegt, die die „*URL* /[Klasse]/[Methode]/[falls nötig, Parameter]“ beinhaltet.

Beispiel:

<form action="<?php echo URL; ?>verwaltung/add\_person" method="POST">

**Controller**

Dieser POST wird vom **Controller** entgegengenommen. Dieser beinhaltet die aufgerufene Methode und führt diese aus. Die Methode im Controller ruft die das zugehörige Model und die dortige Methode mit Parameterübergabe auf.

Beispiel:

<?php

class Verwaltung extends Controller

{

public function add\_person()

{

if (isset($\_POST["submit\_add\_person"]))

{

//Model wird in Variable gespeichert //und ausgeführt

$verwaltungs\_model = $this->loadModel('VerwaltungsModel');

$verwaltungs\_model->add\_person($\_POST["str\_nachname"],

$\_POST["str\_vorname"], $\_POST["d\_gebdatum"], $\_POST["int\_groesse"], $\_POST["str\_bild"], $\_POST["b\_betreuer"], $\_POST["int\_tel"], $\_POST["str\_user"], $\_POST["str\_password"]);

}

header('location: ' . URL . 'verwaltung/index');//Weiterleitung nach

Ausführen der Methode

} } ?>

**Model**

Die Methode im Model führt letztendlich die eigentliche Funktion durch. Am Ende leitet die Methode und damit das Model auf eine andere Seite.

Beispiel:

<?php

class VerwaltungsModel

{

public function add\_person($str\_nachname, $str\_vorname, $d\_gebdatum, $int\_groesse, $str\_bild, $b\_betreuer, $int\_tel, $str\_user, $str\_password)

{

$str\_nachname = strip\_tags(str\_nachname);

$str\_vorname = strip\_tags($str\_vorname);

$d\_gebdatum = strip\_tags($d\_gebdatum);

$int\_groesse = strip\_tags($int\_groesse);

$str\_bild = strip\_tags($str\_bild);

$b\_betreuer = strip\_tags($b\_betreuer);

$int\_tel = strip\_tags($int\_tel);

$str\_user = strip\_tags($str\_user);

$str\_password = strip\_tags($str\_password);

$sql = "INSERT INTO person (name, v\_name, geb\_datum, groesse, bild, betreuer, tel, username, password) VALUES (:name, :v\_name, :geb\_datum, :groesse, :bild, :betreuer, :tel, :username, :password)";

$query = $this->db->prepare($sql);

$query->execute(array(':name' => $str\_nachname, ':v\_name' => $str\_vorname, ':geb\_datum' => $d\_gebdatum, ':groesse' => $int\_groesse, ':bild' => $str\_bild, ':betreuer' => $b\_betreuer, ':tel' => $int\_tel, ':username' => $str\_user, ':password' => $str\_password ));

}}?>