MVC-Framework

Mario Kunz

11.06.2016

Inhalt

[Zweck 3](#_Toc453442928)

[Übersicht 4](#_Toc453442929)

[Module 5](#_Toc453442930)

[Dispatcher 5](#_Toc453442931)

[Das Request Objekt 5](#_Toc453442932)

[Das Response Objekt 5](#_Toc453442933)

[Die Controller Klasse 6](#_Toc453442934)

[Die View Klasse 6](#_Toc453442935)

[Das Model 6](#_Toc453442936)

[Model 6](#_Toc453442937)

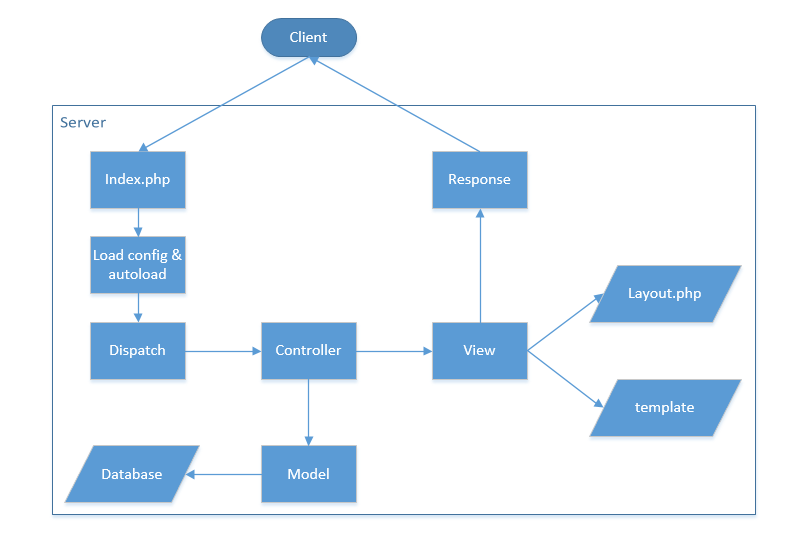
[Aufbau der Models 7](#_Toc453442938)

[Installationsanleitung 8](#_Toc453442939)

# Zweck

Im folgenden Dokument werden die technischen Aspekte des selber entwickelten MVC-Framework genauer Beschrieben und für dritt Anwender verständlich niedergelegt.

# Übersicht



Jede Anfrage auf den Webserver folgt einem strikten Ablauf.

1. Index.php lädt alle erforderlichen Klassen bzw. autoload.
2. Es werden Grundlegende Werte als Konstanten definiert.
3. Die Request-URI wird auseinandergenommen und wird auf den richtigen Controller geleitet (Dispatching)
4. Der Controller verarbeitet die Logik und gibt Werte an die View weiter oder gibt ein Response-Objekt zurück
5. Die View wird zusammengebaut (meistens mittels Layout.php und den templates)
6. Das Response Objekt oder der HTML-Content wird zurückgegeben.

# Module

## Dispatcher

Im Dispatcher wird mittels der ControllerResolver Klasse herausgefunden, welchen Controller gebraucht wird und danach wird auf diesem Controller die richtige Methode aufgerufen.

## Das Request Objekt

Das Request Objekt steht für die komplette Anfrage. Es beinhaltet folgende Werte.

* Uri 🡪 die komplette Request-URI
* Base 🡪 Der komplette Pfad zum Projekt
* Post 🡪 Das ganze $\_POST Array
* Query 🡪 Das ganze $\_GET Array
* Session 🡪 Das ganze $\_SESSION Array
* Params
  + Controller 🡪 Der aktuelle Controller
  + Action 🡪 Die Action, welche im Controller aufgerufen wird
  + Passed 🡪 $\_POST + $\_GET + $\_FILES

Mit der createRequest() Methode wird ein neues Request Objekt anhand der aktuellen Umgebungswerte (Request-URI, $\_GET etc.) kreiert und zurückgegeben.  
URI wird in diesem Zusammenhang auseinandergenommen und der Controller und die Action herausgelesen. Für beides gibt es Standard Werte, wenn sie nicht vorhanden sind.

/acontroller/anaction 🡪 controller => acontroller, action => anaction

/acontroller 🡪 controller => acontroller, action => index

/ 🡪 controller => index, action => index

## Das Response Objekt

Das Response Objekt steht im Gegensatz zum Request Objekt für die Antwort an den User auf dem Client. Das Response Objekt enthält nur die zwei folgenden Werte.

* Headers 🡪 alle gesetzten http-Headers
* Body 🡪 der Inhalt der Response

Über das Response Objekt können Header mit der header(header, value) Funktion der Response übergeben werden. Diese headers werden, dann bevor die Antwort an den Client zurückgeschickt wird, noch gesetzt.

Wenn im Controller mit der redirect Funktion ein Umleiten der http-Anfrage erzwungen wird, wird im Response Objekt ein Location-Header gesetzt und danach wird das ganze Response Objekt an den Controller zurückgegeben.

Sobald in einer Controller Funktion ein Response Objekt zurückgegeben wird, wird das ganze rendern der View übersprungen und der Inhalt der Body Eigenschaft sofort an den Client zurückgesendet.  
Somit ist es sehr einfach möglich auch Bilder, JSON-Strings oder andere MIME-Type Nachrichten zurück an den Client zu senden ohne, dass diese Nachrichten immer in das Layout der Seite gebettet sind.

## Die Controller Klasse

Jeder Controller muss von dieser Superklasse erben.

Die Controller Klasse beinhaltet folgendes.

* Template
* Request-Objekt
* Response-Objekt
* View-Objekt
* Autorender 🡪 sagt ob die Response gerendert werden soll oder nicht.

Das Template muss der gleiche Namen haben, wie die Action des Controllers und muss in einem Unterordner des /View/templates/<Controller>/<action>.php Verzeichnis sein.

Das template kann vom Controller dynamisch während des Ablaufes der Request mittels der \_setTemplate() Methode geändert werden.

Die invokeAction Methode ruft die Action auf dem richtigen Controller ab.

Die Redirect Funktion setzt einen Location Header und setzt die Autorender Eigenschaft auf false. Am Schluss wird das Request Objekt zurückgegeben.

Die Render Funktion holt die Templates und das Layout und fügt alles zusammen. Die http-Headers werden hier gesetzt und das Response Objekt zurückgegeben.

## Die View Klasse

Eigenschaften:

* Template 🡪 Pfad zum Template
* Vars 🡪 alle Variablen, welche später in der View gebraucht werden und vom Controller gesetzt wurden
* Body 🡪 Der body der Response (wird später im Response Objekt gesetzt)
* Content 🡪 Inhalt des Templates
* Request 🡪 Das Request Objekt
* Response 🡪 Das Response Objekt

Mit der Render Methode wird das template ausgeführt und die Ganze View zusammengebaut. Mittels eines Output Buffers (ob\_start) wird alles was geholt wird (template und layout.php) in eine Variable gepackt, welche danach zurückgegeben wird.

Vom Controller aus können mittels der assign Methode Werte an die View übergeben werden, welche dann auf der View angezeigt werden können.

## Das Model

### Model

Die Model Klasse muss wieder von allen Submodel erweitert werden, da sie wichtige Funktionalitäten enthält.

Mittels der readAllArray Funktion kann ein statement Objekt ausgeführt werden und das Resultat als Array zurückerhalten werden. Es kann auch direkt ein Query in Stringform an die Funktion übergeben werden. Wenn kein Resultat von der DB kam wird null zurückgegeben.

Die readAllOrSingle vereinfacht die Funktion der readAllArray, denn wenn hier nur eine Zeile der Tabelle zurückgegeben wurde, wird auch kein mehrdimensionales Array zurückgegeben sondern nur genau die eine Zeile.

Die readAll Methode liest alle Datensatze der aktuellen Tabelle und gibt diese zurück.

Bei der readById wird anhand einer ID die gewünschte Zeile einer gewissen Tabelle ausgelesen.

Die exec funktion ist eine Vereinfachung der execute Funktion von mysqli.

### Aufbau der Models

Ein Model besteht grundsätzlich immer aus einer Table Klasse und einer Entity Klasse.

Ein Entity repräsentiert immer eine Zeile in der Datenbank und beinhaltet so auch alle Attribute, welche in der Tabelle sind.

Die Table Klasse repräsentiert die ganze Tabelle und beinhaltet alle wichtige Funktionen um Datensätze aus der Tabelle zu holen.

Diese Funktionen sind folgende und selbsterklärend.

* create
* readAll
* readById
* update
* delete
* deleteById

Je nach Tabelle sind noch andere Funktionen verfügbar wie zum Beispiel getImagesByGallery.

# Installationsanleitung

Folgend ist eine Installationsanleitung zu meinem Projekt.

1. Ganzes Projekt in einen Ordner kopieren, welcher danach vom Webserver angesprochen werden kann
2. SQL-Skript imgdb.sql laufen lassen.
3. Der Document Root im Webserver **MUSS** auf dem webroot Ordner des Projektes sein sonst funktioniert es nicht.  
   Durch diese Massnahme können die einzelnen php-Files nicht direkt angesprochen werden können.  
     
   Diese 2 Zeilen müssen geändert werden (Pfade noch anpassen):  
   DocumentRoot "E:/IT/xampp/htdocs/imgDB/webroot"

<Directory "E:/IT/xampp/htdocs/imgDB/webroot">

1. Die extension=php\_fileinfo.dll muss in der php.ini aktiviert werden.
2. Unter webroot muss die folgende Ordnerstruktur bestehen.
   * galleries
     + final
     + thumbnail
3. Je nachdem müssen in der config/config.php Datei noch die Datenbank Verbindungsangaben geändert werden.

Bestehender User: [test@gibb.ch](mailto:test@gibb.ch) Passwort: gibbiX12345

Nach diesen Schritten sollte die Applikation funktionieren sonst kontaktieren sie mich unter der folgenden Email Adresse.

[mario99.kunz@gmail.com](mailto:mario99.kunz@gmail.com)