

OOD

Object Oriented Design

Markus Gehrig

# Inhaltverzeichnis

[Inhaltverzeichnis 1](#_Toc16258236)

[1 MVC – Pattern 2](#_Toc16258237)

[1.1 Dispatcher 2](#_Toc16258238)

[1.2 Globale Variablen 2](#_Toc16258239)

[1.3 Methoden Dispatcher 2](#_Toc16258240)

[1.3.1 Nassi Schneidermann Dispatcher 3](#_Toc16258241)

[2 Controller 4](#_Toc16258242)

[2.1 AbstractController 4](#_Toc16258243)

[2.1.1 Globale Variablen 4](#_Toc16258244)

[2.1.2 Methoden AbstractController 4](#_Toc16258245)

[3 Model 4](#_Toc16258246)

[3.1 Modelklasse 4](#_Toc16258247)

[3.2 Repository 4](#_Toc16258248)

[3.2.1 AbstractRepository 4](#_Toc16258249)

[4 Ordnerstruktur 6](#_Toc16258250)

# MVC – Pattern

Für die Umsetzung der Anwendung wird das MVC Pattern benutzt, es ist sehr weit verbreitet und bei Webanwendungen das meist benutzte Pattern.

## Dispatcher

Der Dispatcher nimmt alle Anfragen entgegen und leitet diese an die richtigen Stellen weiter. Er ist also vergleichbar mit dem Telefonisten aus vergangenen Tagen.

Die für seine Aufgabe nötigen Informationen bekommt durch GET-Parameter. Ein normaler Aufbau der URL sieht wie folgt aus:

index.php?**module**=dashboard&**controller**=dashboard&**action**=show&**data**=my-cooldata

|  |  |
| --- | --- |
| **module** | **Modul welches aufgerufen werden soll.** |
| **controller** | **Controller des Moduls welches aufgerufen wird.** |
| **action** | **Aufzurufende Action des Controllers** |
| **data** | **Daten welchen an den Controller weitergegeben werden sollen.** |

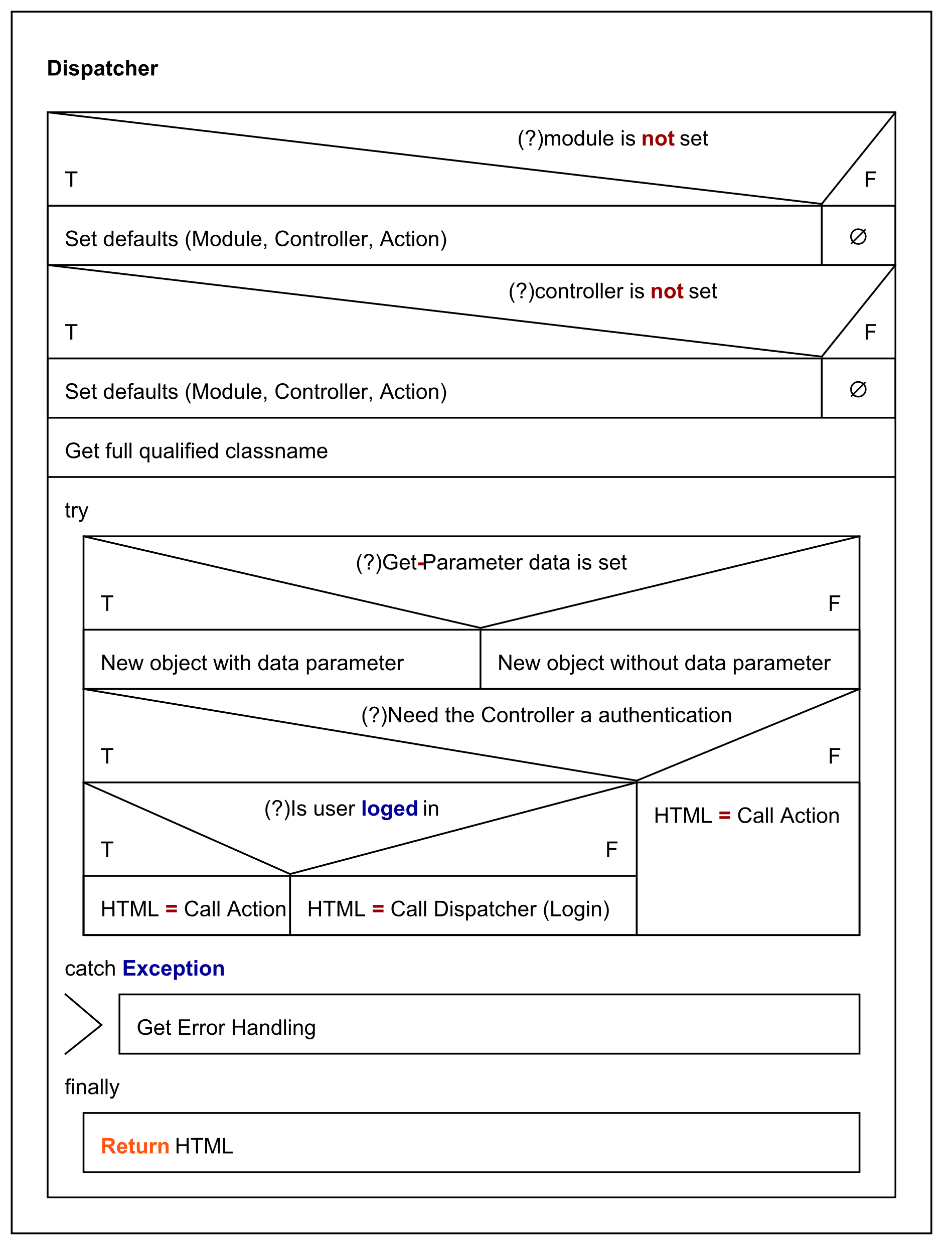
## Globale Variablen

|  |  |  |
| --- | --- | --- |
| module | private | string |
| controller | private | string |
| action | private | string |
| data | private | undefined |

## Methoden Dispatcher

|  |  |  |  |
| --- | --- | --- | --- |
| setModule | public | module:string |  |
| getModule | public |  | string |
| setController | public | controller:string |  |
| getController | public |  | string |
| setAction | public | action:string |  |
| getAction | public |  | string |
| setData | public | data |  |
| getData | public |  | data |
| dispatch | public | module:string controller:string action:string data | string |
| redirect | public | module:string controller:string action:string data |  |
| getClassName | private | module:string controller:string action:string | string |

### Nassi Schneidermann Dispatcher



# Controller

Die Controller sind immer auf ihre jeweilige Ausgabe zugeschnitten. Sie bestehen mindestens aus einem Konstruktor und einer Action.

## AbstractController

Alle Controller erben von der Klasse AbstractController. Die AbstractController Klasse stellt den Controllern Methoden für das Frontendrendering zu Verfügung.

### Globale Variablen

|  |  |  |
| --- | --- | --- |
| templateDirs | protected | array |

### Methoden AbstractController

|  |  |  |  |
| --- | --- | --- | --- |
| addTemplateDir | public | templateDir:string |  |
| setTemplateDirs | public | templateDirs:array |  |
| getTemplateDirs | public |  | templateDirs:array |
| render | public | templateName:string parameters:array | html:string |

# Model

Das PHP Model besteht aus zwei Klassen. Einem Model für jede Datenbank und einem dazugehörigem Repository.

## Modelklasse

Ein Model besteht ausschliesslich aus Variablen und den dazugehörigen Getter und Setter Methoden.

## Repository

Ein Repository selektiert Daten von der Datenbank und erzeugt mit den Daten davon ein Model/mehrere Modelle und gibt diese zurück.

### AbstractRepository

Jedes Repository erbt von der Klasse AbstractRepository. Diese Klasse stellt Funktionen für alle Repositorys zur Verfügung.

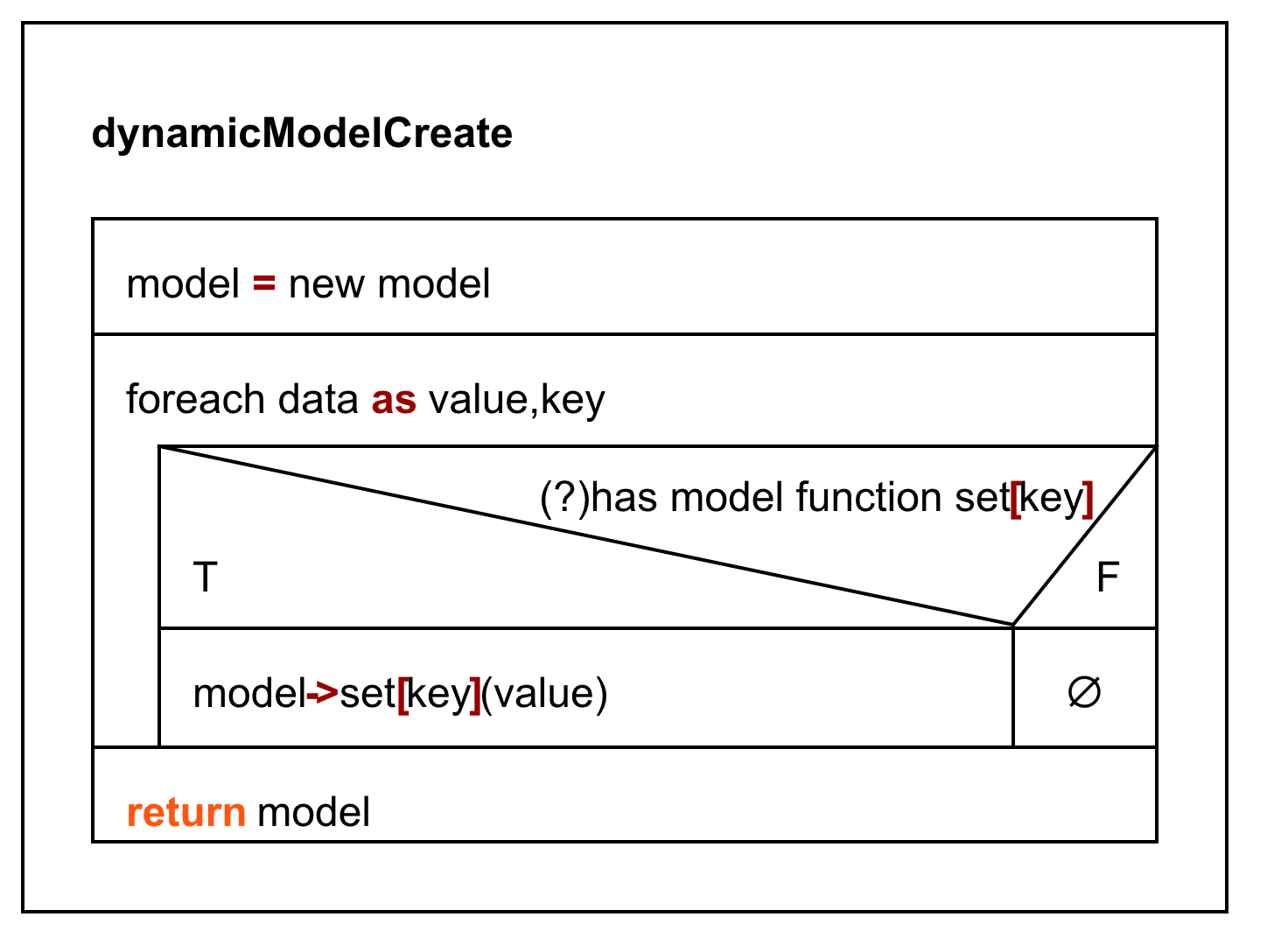
#### Globale Variablen

|  |  |  |
| --- | --- | --- |
| table | protected | string |
| model | protected | string |

#### Methoden AbstractController

|  |  |  |  |
| --- | --- | --- | --- |
| getByUid | public | uid:int | element:object |
| dynamicModelCreate | protected | data:array |  |

#### Nassi Schneidermann dynamicModelCreate



# Ordnerstruktur

Die Ordnerstuktur ist sehr einfach aufgebaut. Es unterscheidet zwischen Core und Modulen, welche die einzelnen Ansichten darstellen.

* root
  + core
    - view
    - dispatcher
    - database
    - controller
  + module
    - dashboard
      * controller
    - …
  + utilities
    - pdf
    - …

|  |  |
| --- | --- |
| **core** | **Globale Klassen, welche häufig gebraucht werden.** |
| view | Klassen welche für das Frontend-Rendering verantwortlich sind. |
| dispatcher | Klassen welche für das Routing der Anfragen verantwortlich sind. |
| **module** | **In sich geschlossene Module (FE-Ansichten)** |
| **utitlities** | **Klassen welche spezielle Funktionen zur Verfügung stellen** |