Inhalt

Abbildungsverzeichnis

Abkürzungsverzeichnis

WAF	Web-Application-Framework
PHP5	PHP: Hypertext Preprocessor 5
LAMPP	Linux-Apache-MySql-PHP-Perl
OSI-Modell	Open-Systems-Interconnection-Modell
HTTP	HyperText Transfer Protocol
WWW	World Wide Web
TCP/IP	Transmission Control Protocol / Internet Protocol
HMVC	Hierarchical Model View Controller
IoC	Inversion of Control
DI	Dependency Injection
PAC	Presentation Abstraction Control
APP	Application
MOD	Modules
LIB	Library
FW	Framework
CRUD	Create Read Update Delete
EXEC	Execute
SQL	Structured Query Language
ACL	Access Control List
URL	Uniform Resource Locator
CPU	Central Processing Unit

1 Einleitung

Die vorliegende Bachelorarbeit befasst sich mit der Konzeption und Erstellung eines Web-Application-Frameworks.

Frameworks sind vorgefertigte Programmiergerüste, welche die Entwicklung der eigentlichen Anwendung beschleunigen sollen und in der Regel vorgefertigte Lösungswege für typische Aufgaben anbieten. Wahrscheinlich jeder Softwareentwickler hat, teilweise unbewusst, schon einmal Frameworks benutzt. So kommen diese bei der Entwicklung von Anwendungen mit grafischen Benutzeroberflächen häufig zum Einsatz (so genannte GUI-Frameworks: z.B. Qt/GtK) und bieten dem Softwareentwickler Komponenten und Abläufe an, um Daten visuell aufzuwerten. [GoF2011 S.36]

Frameworks unterscheiden sich somit von Klassenbibliotheken, da sie nicht nur eine Ansammlung von einzelnen Komponenten sind, sondern zusätzlich auch Strukturen und Abläufe anbieten. [GoF2011 S.36]

Web-Application-Frameworks (im folgenden auch WAF genannt) sind eine Untermenge der Frameworks, die sich auf Anwendungsfälle bei der Entwicklung von Webanwendungen spezialisiert haben.

Die hier behandelte Problemstellung entstand somit aus dem Wunsch heraus, wiederkehrende Probleme beim Erstellen von Webanwendungen effizient lösen zu können. Dabei geht das hier präsentierte Projekt über die Funktionen einer klassischen Programm-Bibliothek hinaus und bietet zusätzlich vorgefertigte Abläufe und Problemlösungen an. Das hierbei entstandene Softwareprodukt steckt somit einen Rahmen ab, in dem Anwendungen vornehmlich mittels Prototyping oder einer anderen (agilen) Entwurfsmethodik implementiert werden können.

Ziel ist es, schneller Web-Applikationen entwickeln zu können, Programmcode zu modularisieren, Handlungsabläufe zu standardisieren und Strukturen zu definieren.

Ein WAF kann somit Projektrisiken und entstehende Projektkosten in Zeit und Geld weiter minimieren und gleichzeitig die Wartbarkeit und Portabilität der entworfenen Module oder Anwendungen erhöhen.

1.1 Motivation

Entwickler von Web-Applikationen begegnen häufig wiederkehrenden Aufgaben. Oftmals werden Methoden und Klassen aus bereits fertigen Projekten in andere Projekte kopiert, um dort ähnliche Funktionalität bereitstellen zu können. Dies kann insbesondere zu Komplikationen führen, wenn der Code in einer schlecht wiederverwendbaren Form vorliegt, projektspezifische Anpassungen enthält oder zusätzliche Abhängigkeiten eingeführt sind. Mit steigender Projektgröße wird so das Projektmanagement erschwert, da das Implementieren von neuen Features gegenüber dem Beheben von Fehlern in den Hintergrund tritt. Dem kann entgegengewirkt werden, indem Software modularisiert und standardisiert wird. Dies leisten sogenannte Web-Application-Frameworks. Sie bieten Bibliotheken und Abläufe an, um typische Probleme im Umfeld von Web-Applikationen zu lösen.

1.2 Gründe für die Neuentwicklung eines Web-Application-Frameworks (WAF)

Nach der Begutachtung bereits existierender Web-Application-Frameworks hinsichtlich deren Funktionalität und deren Grenzen entschied ich mich für den Entwurf eines eigenen Produktes, um meine Softwareprojekte besser realisieren zu können. Dies hat folgende Gründe:

- **Komplexität und Overhead:** Viele Frameworks sind auf Enterprise-Applikationen ausgelegt und somit für die meisten Einsatzgebiete unnötig komplex. Mit einem eigenen Entwurf kann das spezifische Einsatzgebiet passgenau abgedeckt werden, ohne dass die Software überflüssige Funktionen enthält.
- **Paradigmenzwang:** Die größte Stärke eines Frameworks, einen Rahmen vorzugeben, kann auch gleichzeitig seine größte Schwäche sein: wenn nämlich die zur Verfügung gestellten Komponenten nicht das Erforderliche leisten. Bereits existierende, teils komplexe Automatismen zu ergänzen, ohne die systemweiten Auswirkungen vollständig zu kennen, kann jedoch zu schwer vorhersehbarem Verhalten führen.
- **Abhängigkeit:** Die Nutzung eines Fremdproduktes schafft natürlich auch Abhängigkeiten zu diesem. So muss auf das Erscheinen neuer Funktionen und auf die Behebung von Fehlern gewartet werden. Gleichzeitig können Sicherheitslücken bis zur Bereitstellung eines Updates entsprechend lange ausgenutzt werden.

So bieten die auf dem Markt verfügbaren Frameworks zwar für alle typischen Probleme Lösungswege an, jedoch unterscheiden sich diese Lösungswege teilweise deutlich voneinander: Sie sind abhängig von der eingesetzten Umgebung (z.B. Windows/Linux), der eingesetzten Programmiersprache (z.B. Ruby/Perl/Java/PHP/node.js), der Projektgröße (z.B. enterprise/middle/small), dem speziellen Einsatzgebiet (z.B. Frontend/Backend/Echtzeit/Domain/...), den zusätzlich zur Verfügung gestellten Funktionen sowie der Strukturierung der Komponenten.

Ich stand vor der Notwendigkeit, ein neues WAF zu entwickeln, da mein gewünschtes Einsatzgebiet unzureichend abgedeckt war: Viele nützliche Features, z. B. Abfragen zu kaskadieren, stehen nur in Enterprise-WAFs zur Verfügung, obwohl auch kleinere und mittlere Projekte davon profitieren würden. Für die Entwicklung meiner Software ist mir die Lauffähigkeit in einer LINUX-Apache-MySQL-PHP-Perl (im folgenden auch LAMPP genannt) Umgebung wichtig, da diese eine weit verbreitete entwicklerfreundliche Plattform mit hoher Kundenakzeptanz ist. Als Entwickler von auf spezielle Kundenwünsche gefertigter Individualsoftware sollte das WAF klar strukturiert sein und mit möglichst wenig Komplexität vorliegen. Dazu trägt die Zentralisierung des Ein- und Ausstiegspunkts bei, um z.B. eine vorgelagerte Datenfilterung zu ermöglichen. Zudem ist die Zentralisierung des Objektmanagements praktikabel, um das Erzeugerverhalten bezüglich verschiedener Objekte vereinheitlichen zu können. Desweiteren sollte die Möglichkeit bestehen, Anfragen untereinander zu verschachteln, um die Komponentenbildung zu vereinfachen.

1.3 Häufig genutzte WAF

Die meisten Frameworks im LAMPP-Umfeld beherrschen z.B. nicht die Fähigkeit, Anfragen zu verschachteln. Sie bieten dafür jedoch Behelfsfunktionen, die dieses Feature aber höchstens teilweise nachrüsten können. Oftmals bestehen Anwendungen jedoch aus dem Zusammenspiel von mehreren Einzelkomponenten, z.B. einer "Wer-ist-online-Box" im linken Teil einer Website, dem Seiteninhalt aus einer Datenbank in der Mitte und einer Anzeige von zufällig ausgewählten Partnern im rechten Teil der Seite. In diesem Beispiel könnten bequem 3 unabhängige Einzelkomponenten über eine verschachtelte Anfrage zusammengesetzt werden. Dieses Vorgehen erhöht die Modularisierung und damit die Wiederverwendbarkeit des Quellcodes immens. Trotzdem unterstützen die meisten WAF wie CakePHP, CodeIgniter, Symfony, APF, FLOW3 oder ZendFramework dieses Vorgehen nur unzureichend. Einzig Kohana bietet dieses Feature, ist jedoch recht komplex in Umfang und Bedienung und trifft damit nicht das vorgestellte Einsatzgebiet.

1.4 Zielsetzung und Einsatzbereich

Bei der Entwicklung ging es nicht nur um die Konzeption und Implementation eines Web-Application-Frameworks im Linux-Apache-MySQL-PHP-Perl-Umfeld, um Applikationen zielorientierter entwickeln zu können. Es sollten auch tiefgreifenden Kenntnissen in diesem Bereich erlangt werden. Die Praxistauglichkeit des entworfenen WAF sollte zudem an einer Beispielapplikation demonstriert werden.

Das Projekt setzt eine typische und sehr weit verbreitete Umgebung für Web-Applikationen voraus, die meist aus einem Linux-Server mit installiertem PHP5.x, einer Datenbank und einem Webserver besteht. Das Projekt soll vornehmlich bei kleineren und mittleren Anwendungen zum Einsatz kommen. Enterprise-Funktionalität bleibt damit unberücksichtigt. Diese Art von Software kann somit als "micro-Framework" bezeichnet werden.

Ein Web-Application-Framework hat die Aufgabe, Softwarekomponenten zu standardisieren und Lösungsansätze für typische Anwendungsfälle im Bereich Web-Applikationen zu bieten, insbesondere:

- Routing von ankommenden Anfragen
- Verwalten von Daten aus einer Datenbank
- Darstellung und Manipulation von Daten
- Trennung von Programmlogik und Datendarstellung
- Verwaltung und Konfiguration von Objekten
- Modularisierung von Applikationen

Die im Folgenden vorgestellten Techniken und Entwurfsmuster stellen einen Weg dar, um diesen Anforderungen im Rahmen der Möglichkeiten der zugrundeliegenden Anwendungen und Protokolle gerecht zu werden.

2 Grundlagen

Das Verstehen der Thematik setzt gewisse Grundkenntnisse voraus. Im folgenden Kapitel werden einige später angewandte Techniken kurz erläutert sowie Grenzen abgesteckt.

Um die Interoperabilität zwischen Computern zu gewährleisten und somit überhaupt Datenaustausch zu ermöglichen, bedarf es einer Fülle von Protokollen und Standards. Dies beginnt bei der Definition des Übertragungsmediums, der Topologie des Netzes, der benötigten Hardware und führt von grundlegenden Regeln der Kommunikation wie Kollisionsvermeidungsalgorithmen, Paket- oder Leitungsvermittlung, Zielfindung der Daten, Fehlererkennungs- und Korrekturverfahren zu den zweckspezifischen Protokollen und letztendlich zur Anwendung selbst.

Das *OSI-Modell* mit den darin untergebrachten und von der Hardware, vom Betriebssystem und den entsprechenden Servern implementierten Standards sorgt als Middleware für die Bereitstellung und Abstraktion der notwendigen Dienste, um eine Kommunikation überhaupt zu ermöglichen. Auf all diesen essentiellen Techniken baut die hier vorgestellte Applikation auf. Es würde jedoch den Rahmen dieser Publikation sprengen, auf all diese Grundlagen einzugehen. Dank der entkoppelnden Eigenschaft des *OSI-Modells* reicht es, auf die direkt darunterliegende Schicht einzugehen: *HTTP*. [11]

2.1 Begriffsdefinitionen

Im Folgenden werden die für das Verständnis der Arbeit wesentlichen Begriffe kurz erklärt. Die hier vorgestellten Definitionen sind ein Auszug aus dem unten angefügten Glossar (*kursiv* markiert) und sollen ein grundlegendes Verständnis über die in der Arbeit verwendeten Fachtermini schaffen.

HTTP(Hypertext-Transfer-Protocol) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten aus dem World Wide Web (*WWW*) in einem Webbrowser darzustellen. Das Protokoll gehört der Anwendungsschicht im *OSI-Schichtenmodell* an. Die Anwendungsschicht wird von den Anwendungsprogrammen angesprochen, im Fall von *HTTP* ist das meist ein Webbrowser oder ein Webserver. *HTTP* ist ein zustandsloses Protokoll. Durch Erweiterung seiner Anfragemethoden, Header-Informationen und Statuscodes ist *HTTP* nicht auf Hypertext beschränkt, sondern wird zunehmend zum Austausch beliebiger Daten verwendet. Zur Kommunikation ist *HTTP* auf ein zuverlässiges Transportprotokoll angewiesen, wofür in nahezu allen Fällen *TCP* verwendet wird. [1]

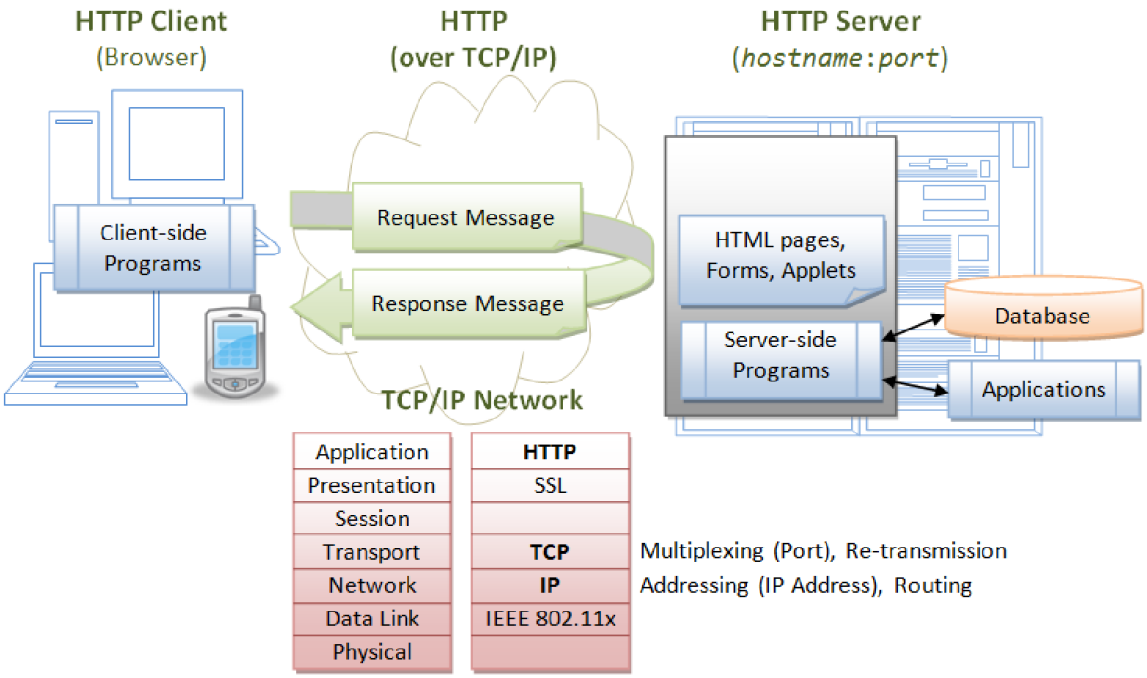


Abbildung 1: Übersicht über die Bestandteile von HTTP

Software-Frameworks sind Halbzeuge, die eine wiederverwendbare, gemeinsame Struktur einer Endanwendung zur Verfügung stellen. Ein Framework gibt somit in der Regel die Anwendungsarchitektur vor. Dabei findet meist eine Umkehrung der Steuerung (*Inversion of Control*) statt: Der Programmierer registriert konkrete Implementierungen, die dann durch das Framework gesteuert und benutzt werden, statt – wie bei einer Programmbibliothek – lediglich Klassen und Funktionen zu benutzen. Wird das Registrieren der konkreten Klassen nicht fest im Programmcode verankert, sondern "von außen" konfiguriert, so spricht man auch von *Dependency Injection*. Die Entkopplung der Datenschicht von der Präsentationsschicht wird meist durch die Entwurfsmuster *(H)MVC* oder *PAC* geleistet. Ein solches Framework sorgt somit dafür, dass man sich nur noch auf die Entwicklung der Anwendung konzentrieren muss. [2; GoF2011, S.37]

Entwurfsmuster sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme in der Softwarearchitektur und -entwicklung. Sie bieten erprobte, universelle Konzepte, die oftmals über das direkt offensichtliche Problem hinausgehen. [3; GoF2011, S.3]

Prototyping ist eine Methode der Softwareentwicklung, die schnell zu ersten Ergebnissen führt und frühzeitiges Feedback bezüglich der Eignung eines Lösungsansatzes ermöglicht. Dadurch ist es möglich, Probleme und Änderungswünsche frühzeitig zu erkennen und mit weniger Aufwand zu beheben, als es nach der kompletten Fertigstellung möglich ist. Unterschieden wird zwischen explorativem, evolutionärem und experimentellem *Prototyping*. Jeder Art des Prototypenbaus ist jedoch gemein, dass dem Kunden kurzfristig ein lauffähiges (Teil-)Programm vorgeführt werden kann und damit eine bessere Kundenkommunikation ermöglicht wird. Fehlplanungen können damit rechtzeitig entdeckt und verhindert werden. [4; GoF2011 S.434]

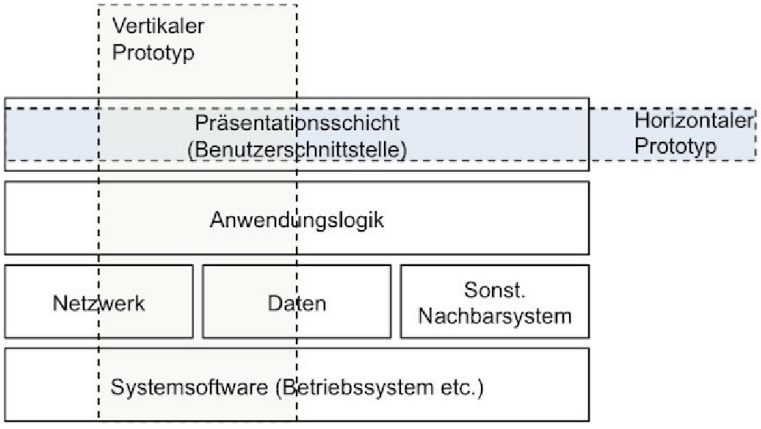


Abbildung 2: Verschiede Strukturen von Prototypen

Coding-Conventions sind Richtlinien, in denen Quellcode zu verfassen ist. Dies beinhaltet unter anderem das Encoding der Datei, Tabulatoreneinstellung und verschiedene Arten der Strukturierung des Quellcodes (z.B. Einrückung, Klammerung, Doc-Blocks) sowie Namenskonventionen (z.B. Schreibweise, CamelCase, treffende Bezeichnungen). Dies vereinheitlicht die Darstellung und erhöht somit das Verständnis des Quellcodes. [5]

Paradigmen sind grundsätzliche Denkansätze. Sie dienen als Hilfsmittel, um falsche Entscheidungen im Designprozess zu vermeiden. Typische Paradigmen sind: "DRY – Don't Repeat Yourself", "KISS - Keep it Stupid and Simple", "SoC - Separation of Concerns", "YAGNI - You Ain't Gonna Need It" und "Convention over Configuration".

Object-Lifecycle-Management bezeichnet eine Art der Ressourcenverwaltung und wie Objekte erschaffen, gelagert und zerstört werden. Dies beeinträchtig die Art wie auf Objekte zugegriffen wird oder wie sie Konfiguriert werden. Typische Entwurfsmuster sind in diesem Bereich der sog. *Builder* oder die *Fabrik*. Ein *Builder* kann somit zentral zum Bauen und Konfigurieren von beliebigen Klassen herangezogen werden und sich zusätzlich um das Verwalten von *Singletons* kümmern.

Template ist eine Schablone, die mit Inhalt gefüllt werden kann. Daten können mittels einer solchen Vorlage formatiert ausgegeben werden. Da in dem Template alle Formatierungsanweisungen stehen, wird das Vermischen von Darstellung und Logik vermieden. Das Design von Benutzeroberflächen kann somit (fast) beliebig verändert werden, ohne andere Programmteile manipulieren zu müssen. Templates werden somit oft als Datenansicht verwendet. [Fowler2009, S.386]

DataMapper/Fachklassen sind zusätzliche Klassen zwischen dem *Controller* und der Datenbank und stellen ein simples *objektrelationales Mapping* mit der Datenbank zur Verfügung. Damit kann applikationsspezifische Logik weitestgehend aus dem *Controller* entfernt werden, so dass der *Controller* nur zum Koordinieren der Anwendung zuständig ist. [Fowler2009, S.189]

FrontController dient als zentraler Einstiegspunkt in eine Anwendung. Mögliche Aufgaben sind: Routing, Filterung der Anfrage, Bestimmung des zu verwendeten *Controllers/Views*. [Fowler2009, S.380]

Inversion of Control (IoC) bezeichnet ein Paradigma, bei dem der Kontrollfluss bewusst und gezielt an einen Automatismus abgegeben wird. Statt dass die Anwendung selbst den Kontrollfluss steuert, steuert das Framework den Ablauf. Die Anwendung greift lediglich an den nötigen Stellen ein. Typische Beispiele sind *Plugin-Systeme*, *Hooks*, *Callbacks* und *Listener*. Oftmals geht *IoC* direkt mit *DI* einher. [6]

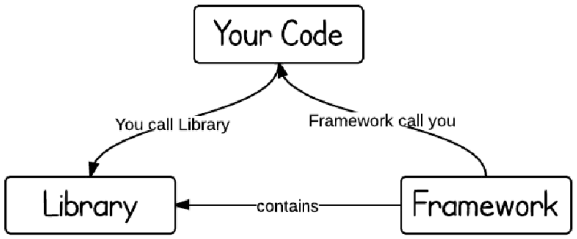


Abbildung 3: Schematische Darstellung von Inversion of Control

Dependency Injection (DI) ist ein Konzept, welches die Abhängigkeiten eines Objekts zur Laufzeit reglementiert. Benötigt ein Objekt bei seiner Initialisierung ein anderes Objekt, ist diese Abhängigkeit an einem zentralen Ort hinterlegt – es wird also nicht vom initialisierten Objekt selbst erzeugt. Das Objekt wird initialisiert und anschließend werden die Abhängigkeiten via Konstruktor oder setter-Methoden übergeben. Eine erweiterte Form von *Dependency Injection* stellt die Verwendung von *Containern* dar. In einem *Container* werden zentral Konfigurationen sowie die benötigten Abhängigkeiten hinterlegt. Soll ein neues Objekt erzeugt werden, wird es nicht direkt selbst erzeugt, sondern durch den *Container*. Dieses hier gezeigte Verfahren ermöglicht *loose-coupling*, erhöhte Testbarkeit sowie eine Zentralisierung von Konfiguration und Abhängigkeit. [7]

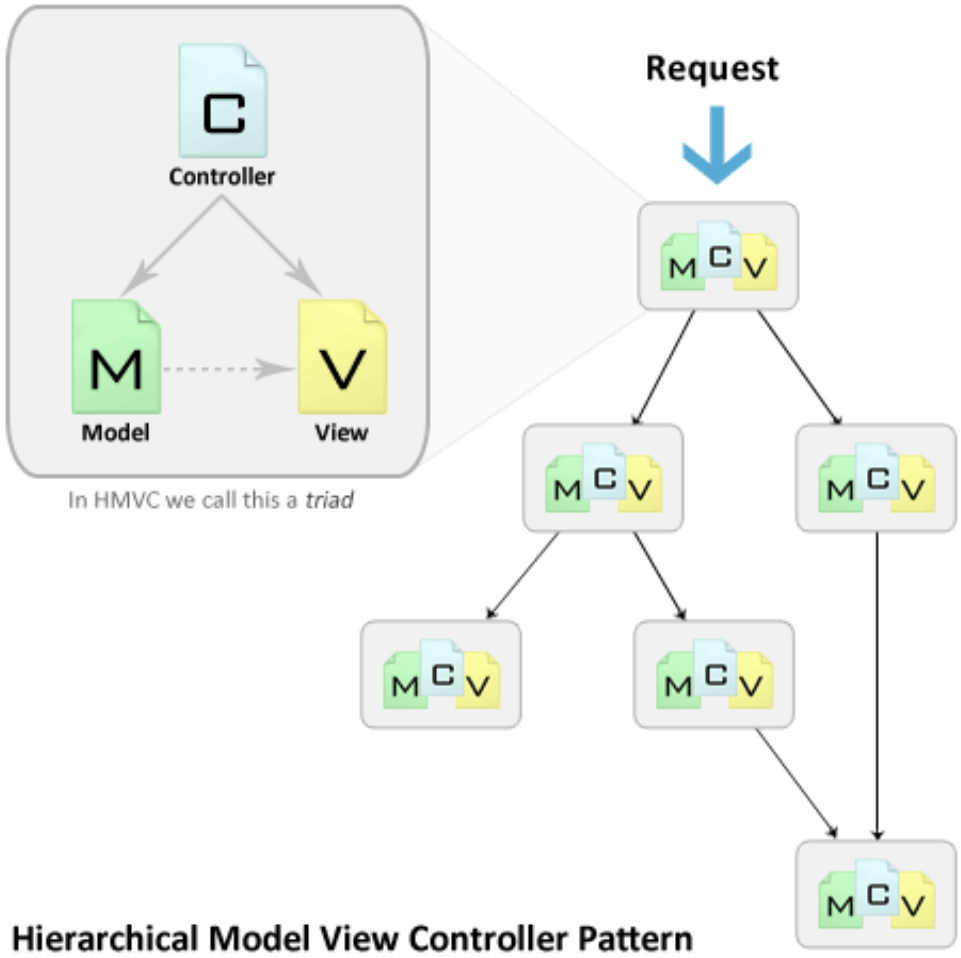
```
private $_preFilter  = null;

private $_postFilter = null;

/** NO DEPENDENCY INJECTION (BAD!) **/
public function __construct() {
    $this->_preFilter  = new FW_Filter_Chain();
    $this->_postFilter = new FW_Filter_Chain();
}

/** CONSTRUCTOR BASED DEPENDENCY INJECTION **/
public function _construct(FW_Filter_Chain $pre, FW_Filter_Chain $post) {
    $this->_preFilter  = $pre;
    $this->_postFilter = $post;
}

/** SETTER BASED DEPENDENCY INJECTION **/
public function setPreFilter(FW_Filter_Chain $preFilter = null) {
    $this->_preFilter  = $preFilter;
}
public function setPostFilter(FW_Filter_Chain $postFilter = null) {
    $this->_postFilter = $postFilter;
}
```

Abbildung 4: Dependency Injection am Quellcodebeispiel

Container versorgen die Komponente mit allen benötigten Schnittstellen zu anderen Systemen. Sie dienen der Komponentenkapselung. Dies dient der Verringerung von Abhängigkeiten (*loose coupling*). Lokale Änderungen lassen sich so ohne Beeinflussung anderer Komponenten durchführen. Container sind ein typischer Einsatzbereich von *Dependency Injection* und *Inverson of Control*. [8]

HMVC *Hierachical-Model-View-Controller* trennt die Darstellung von der Datenhaltung und ermöglicht damit eine Entkoppelung der Geschäftslogik von der Präsentation der Daten. Die entstandenen Schichten können problemlos gegen andere ausgetauscht werden, so dass es möglich ist, ohne viel Aufwand die Datenbank zu wechseln oder die Ausgabe der Daten anzupassen. Beim *HMVC* können diese Komponenten zusätzlich noch verschachtelt werden. Dies erlaubt den Aufbau einer Applikation aus mehreren einzelnen Komponenten. [9]

Abbildung 5: Schematische Darstellung des HMVC-Entwurfsmusters



2.2 Beschreibung der verwendeten Entwurfsmuster

Im Folgenden werden die teilweise oben genannten *Entwurfsmuster* ausführlicher vorgestellt. So wird kurz erklärt welche Vorteile das Entwurfsmuster (im folgenden auch *Pattern* genannt) hat, wie es strukturiert ist und wo es eingesetzt wird.

2.2.1 Model-View-Controller (MVC)

Dieses wohl am häufigsten benutzte *Entwurfsmuster* sorgt dafür, die Anwendung in 3 Bereiche aufzuteilen:

- Das Datenmodell (im folgenden auch *Model* genannt), welches sich um die Datenhaltung kümmert. Dies kann z.B. das Lesen und Schreiben in eine Datenbank oder Datei sein.
- Die Ansicht (im folgenden auch *View* genannt), welche sich um die Darstellung der Daten kümmert. Im Falle eines *WAF* werden die meisten Daten in HTML-Code eingebettet ausgegeben.
- Der *Controller* verbindet das Datenmodell und die Ansicht. Er beinhaltet die eigentliche Applikationslogik.

Je nach Umfang der Anwendung kann es eine sinnvolle Erweiterung sein, die Applikation vom *Controller* zu trennen und in eine extra Klasse zu legen. Diese Klasse wird häufig als *Fachklasse* bezeichnet und kann als Vermittler (z.B. mit einem *Mediator-Pattern*) zwischen *Controller* und *Model* gesehen werden.

Da die Darstellung der Ansicht erst auf dem Computer des Besuchers der Website (im folgenden auch *Client* genannt) erfolgt, gibt die Ansicht meist Text (HTML-Code) zurück. Durch die Dreiteilung der Anwendung können aber auch andere Models oder Views genutzt werden, um z.B. Daten als PDF auszugeben. Dies ermöglicht einen einfachen Austausch der Schichten und eine Trennung von Datenhaltung und Datendarstellung. In Abbildung 5 wird die Struktur schematisch dargestellt. [GoF2011, S.5]

2.2.2 Hierarchical-Model-View-Controller (HMVC)

Dieses *Entwurfsmuster* erweitert das *MVC-Pattern* dahingehend, dass einzelne *MVC-Patterns* als Kaskade dargestellt werden können. Somit kann die Anwendung weiter Modularisiert werden. Ein Verbund aus den *MVC-Komponenten* wird auch *HMVC-Triade* genannt. Eine Anfrage kann somit aus mehreren *HMVC-Triaden* bestehen. Die Aufrufe müssen so gestaltet werden, das sie untereinander isoliert ablaufen. Dies wird durch Kapselung in einem *HMVC-Request-Objekt* erreicht, welchem die zum Ausführen der Anfrage nötigen Parameter übergeben werden. Das *HMVC-Request-Objekt* kümmert sich dann um die Abarbeitung und gibt einen Rückgabewert an den übergeordneten *Controller* zurück. Dies schafft eine einheitliche Schnittstelle (auch Interface genannt) mit der *Controller* aufgerufen werden können. Gleichzeitig wird ein zentraler Anlaufpunkt geschaffen, um Benutzerrechte zu prüfen und das Routing durchzuführen. In Abbildung 5 wird die Struktur schematisch dargestellt. [9]

2.2.3 Front-Controller

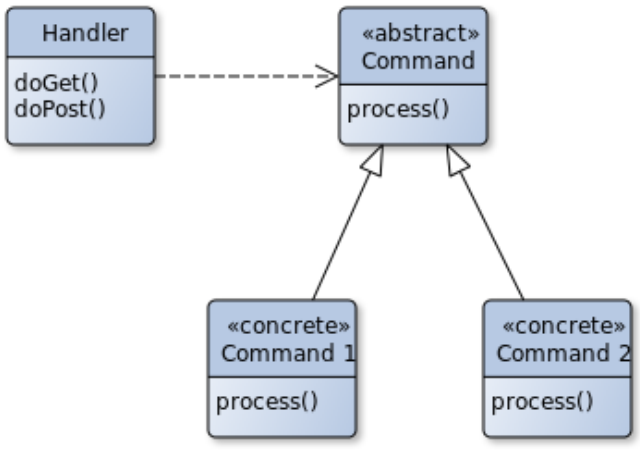


Abbildung 6: Aufbau eines FrontControllers

Der *Front-Controller* kann als zentraler Anwendungseinstiegspunkt betrachtet werden. Er filtert die Ein- und Ausgaben und erstellt die erste *HMVC-Triade*. Er kann desweiteren zusätzliche Aufgaben übernehmen, die vor dem Aufruf des ersten *Controllers* durchgeführt werden müssen. Dies kann z.B. die Benutzerauthentifikation, das Festlegen der Anwendungssprache oder Caching sein. Generell sollten alle Vorbereitungen, die vor dem Aufruf der ersten *HMVC-Triade* erledigt sein müssen, im *Front-Controller* erledigt werden. [Fowler2009, S.380]

2.2.4 Erbauer (Builder)

Dieses Pattern dient dazu, den Erzeugungsprozess von Objekten zu zentralisieren. Ein *Erbauer* kreiert somit ein gewünschtes Objekt von einer beliebigen Klasse. Den Klassen muss jedoch gemein sein, das sie mit dem selben abstrakten Bauplan erbaut werden können. Mit dem Erbauer können z.B. Konfigurationen oder benötigte Objekte (Abhänigkeiten, *Dependency Injection*) den zu erzeugenden Objekten übergeben werden.

Weiterführende Formen des *Erbauer-Patterns* erlauben, verschiedene abstrakte Baupläne zu hinterlegen, um den Bauprozess an die zu erbauenden Objekte anpassen zu können (siehe Abbildung 7). [GoF2011, S.119]

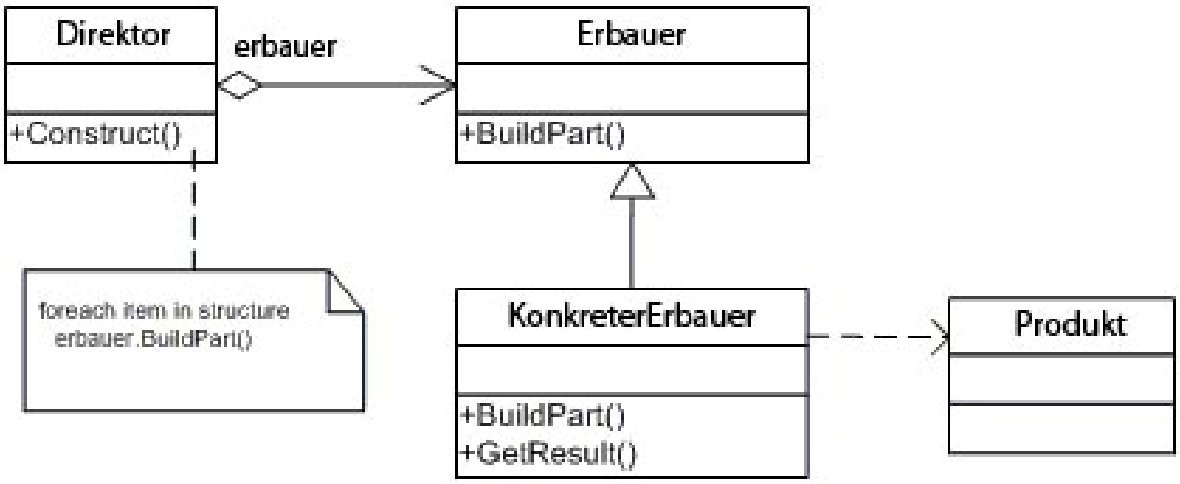


Abbildung 7: Aufbau des Erbauer-Patterns

2.2.5 Singleton (Einzelstück)

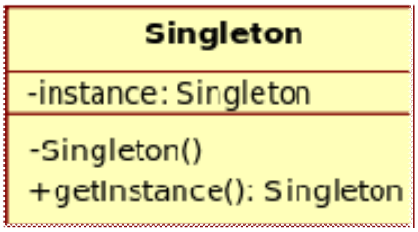


Abbildung 8: Struktur eines Singletons

Der *Singleton* ermöglicht es von einer Klasse nur eine Instanz zu bekommen. Er ähnelt einer Fabrik, mit dem Unterschied, dass er immer das gleiche Objekt zurückgibt, anstatt neue zu erschaffen.

Das Entwurfsmuster limitiert somit die Anzahl der verfügbaren Instanzen auf 1. Soll es mehr als ein Objekt von einer Klasse geben, die Stückzahl aber weiterhin limitiert bleiben, spricht man von einem *Multiton*. Es ist sinnvoll von manchen Objekten nur ein Exemplar zu haben. So gibt es in den meisten Systemen nur einen Benutzer oder einen FrontController, eine Datenbankverbindung, eine HTTP-Anfrage oder eine Antwort. Da der "Konstruktor" des Singletons eine Klassenmethode ist, kann zentral und überall auf die Instanz zugegriffen werden. Der Singleton hat jedoch auch architektonische Nachteile und wird gerne als Anti-Pattern angeführt, da sich Klassen mit Ihren Instanzen schwerer testen lassen. Trotzdem kann es von Vorteil sein Singletons zu verwenden, z.B. bei anderen Erzeugermustern wie dem Erbauer. [GoF2011, S.157]

2.2.6 Composite-Pattern (Kompositum)

Das Composite ist ein Strukturpattern und gliedert ähnliche Objekte in einer baumartigen Struktur. Alle Objekte eines Kompositums implementieren ähnliche Funktionalität mittels einer einheitlichen Schnittstelle. Das *Kompositum* selbst kann weitere *Composite-Patterns* aufnehmen, da auch dort die einheitliche Schnittstelle genutzt wird. Das Pattern kann genutzt werden, um hierarchische Strukturen abzubilden, wie z.B. Verzeichnisstrukturen. Es kann auch Objekte enthalten, über welche iteriert werden kann. Es ist somit möglich es auf Daten anzuwenden. [GoF2011, S.239]

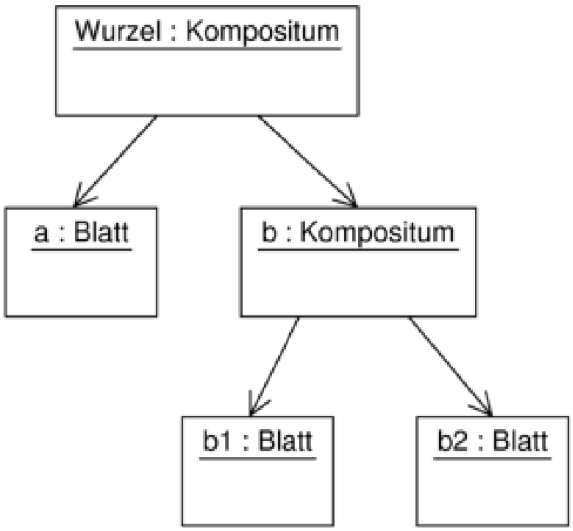


Abbildung 9: Objektdiagramm eines Kompositums

Im hier vorgestellten Framework wird das Pattern verwendet, um die Ein- und Ausgaben zu filtern. Die Eingabedaten durchlaufen somit das *Composite-Pattern* und werden in den einzelnen Blättern manipuliert.

2.2.7 Template-Engine

Eine Template-Engine ist an sich kein Entwurfsmuster, sondern eine Möglichkeit eine Datenansicht umzusetzen. Ziel dabei ist es, die vom *MVC-Pattern* erreichte Separierung von Daten und deren Darstellung weiter zu entkoppeln. Der *View* lädt dabei eine Datei (das sogenannte Template), die lediglich Formatierungsanweisungen, statischen Text sowie Platzhalter enthält und ersetzt diese. Die dazu benötigten Daten erhält er z.B. vom Controller übergeben. Das Template enthält in den meisten Fällen HTML-Code aber auch andere Formate wie TeX, CSV sind denkbar. [Fowler2009, S.386]

```
$content = FW_ObjectManager::createPrototype('LIB_View_HTML', '_index.tpl');
$content->set('body_text', 'Hallo Welt');
$content->set('test_array', array('foo' => 'bar', 'baz' => 'bar'));

echo $content->execute();
```

In diesem Beispiel wird ein Objekt vom Typ "LIB_View_HTML" erzeugt, welche als Parameter einen Dateinamen übergeben bekommt. In dieser Datei stehen die Formatierungsanweisungen und Platzhalter. Anschließend werden dem Objekt die benötigten Daten zum Füllen der Platzhalter übergeben. Zuletzt werden die Platzhalter durch die übergebenen Daten ersetzt.

Beispiel für den Inhalt der Datei "_index.tpl":

```
<h1><?=$body_text; ?></h1>

<? foreach($test_array as $key => $value) : ?>
   <p>Schlüssel: <?=$key; ?> - Wert: <?= $value; ?></p>
<? endforeach; ?>
```

Aus diesen Daten könnte folgende Ausgabe generiert werden:

```
<h1>Hallo Welt</h1>

   <p>Schlüssel: foo - Wert: bar</p>
   <p>Schlüssel: baz - Wert: bar</p>
```

3 Anforderungsanalyse für ein WAF

Aus den oben genannten Eigenschaften und Funktionen kann auf die Anforderungen geschlossen werden, die an ein Web-Application-Framework gestellt werden müssen. Die folgende Auflistung beschreibt kurz die notwendigen Funktionalitäten.

3.1 Funktionale Anforderungen

Initialisierung/Bootstrap

Bootstrapping bezeichnet das "Hochfahren" des Systems. Dateien werden geladen, Konfigurationen eingebunden, Variablen initialisiert. Das System wird in einen Zustand versetzt in dem es arbeitsfähig ist. Daraus ergeben sich folgende Anforderungen:

Das System soll:

- einen Einstiegspunkt zur Verfügung stellen, um weitere systemkritische Komponenten nachladen zu können,
- zentral globale Konstanten und Funktionen definieren,
- Objekte mit ihren entsprechenden Konfigurationen initialisieren und
- die Möglichkeit bieten, entwicklungsumgebungsspezifische Parameter zu setzen.

Dateisystem

Die Struktur im Dateisystem sollte der Struktur der im Framework verbauten Softwarekomponenten entsprechen, um von der verwendeten Klasse direkt auf den Ort im Dateisystem schließen zu können. Aus dem verwendeten *HMVC-Pattern* kann man somit auf eine Zergliederung in *Controller/Model/View*-Klassen schließen, die bereits oben angesprochene Fachklasse wie auch eine Konfigurationsdatei kann zusätzlich implementiert werden. Diese Daten sind in einem Ordner, der dem Anwendungsnamen entspricht, abzulegen. Je nach Art der Anwendung wird der Ordner im Anwendungsverzeichnis oder im Modulverzeichnis abgelegt. Module sind wiederverwendbare Funktionalitäten (Hilfsanwendungen), die von Anwendungen verwendet werden können, wie z.B Seiten-Templates oder ein Modul zum Beschneiden von Bildern.

Daraus ergeben sich folgende Anforderungen:

- Das Dateisystem soll einen applikations- , modul- und frameworkspezifischen Bereich aufweisen, sowie einen Bereich zum Anlegen einer Software-Bibliothek.
- Ein Modul oder eine Applikation soll immer aus einer Konfiguration, einem Controller und ggf. einer Fachklasse sowie entsprechenden Ansichten bestehen.
- Applikationen sollen komplette anwendungsspezifische Funktionen bündeln und von außen direkt ansprechbar sein.
- Module sollen komplette häufig benötigte Komponenten bündeln. Diese Hilfsanwendungen sollten jedoch nicht direkt von außen ansprechbar sein, da sie ihre Funktionalität anderen Anwendungen zur Verfügung stellen.
- Die Software-Bibliothek soll einer typischen Bibliothek entsprechen und bündelt häufig benötigte, nicht frameworkspezifische Funktionen.

Automatisches Laden von Dateien

Der Autoloader-Mechanismus erlaubt es, zur Laufzeit angeforderte Klassen dynamisch nachzuladen. Damit wird das Laden von nicht benötigten Klassen vermieden, gleichzeitig kann der Autoloader Klassennamen in Pfade umsetzen.

- Der Autoloader-Mechanismus soll den Klassennamen in Dateisystem-Pfade auflösen und die zur Laufzeit benötigten Klassen nachladen.

Zentraler Einstiegspunkt

Ein zentraler Einstiegspunkt ermöglicht Datenfilterung und Anfragenverarbeitung an einer Stelle. Hier kann Logik abgelegt werden, die vor jeder Anfrage ausgeführt werden soll. *Front-Controller* sind typische *Patterns* für diese Anforderungen.

Der zentrale Einstiegspunkt soll:

- in der Lage sein, über Filter die Ein- und Ausgabe zu manipulieren,
- nach Bedarf erweiterbar sein,
- die Ursprungs-HMVC-Anfrage erzeugen und die Anfrageparameter an diese weiterleiten und
- das Ergebnis der Anfrage zurückgeben.

Zentrales Objektmanagement

Ein zentrales Objektmanagement erlaubt es, Objekte nach einem Schema zu erbauen und zu verwalten. Aufgaben, die sonst für jedes Objekt einzeln erledigt werden müssten, können so zentralisiert werden. Das *Erbauer-Pattern* ist ein typischer Vertreter um diese Problemstellung zu lösen.

- Das zentrale Objektmanagement soll *Prototypen* und *Singletons* erzeugen und diese mit der Klassenkonfiguration versehen.
- Es soll möglich sein, zentral Objekte abzulegen.

Zentrale Konfiguration

Eine Konfigurationsklasse dient dem Laden von Einstellungen aus Dateien und als zentraler Speicherort für diese. Klassen können dort die zur Laufzeit benötigten Informationen finden.

Die zentrale Konfiguration soll:

- Variablen und Objekte zentral und global anlegen,
- Konfigurationsdateien einlesen, um Klassen konfigurierbar zu machen und
- Konfigurationen zentral lagern.

Routing

Routing beschreibt ähnlich wie in der Netzwerktechnik die Wegfindung. Bei Frameworks ist damit allerdings das Auflösen von URLs in "*HMVC-Pfade*" gemeint.

So könnte die Adresse `http://www.example.com/events/view/14` in folgenden *HMVC-Pfad* aufgelöst werden: `app/Events/Controller/view/14`. `"app"` ist dabei der Anwendungsbereich, `"Events"` die gewünschte Anwendung aus dem Anwendungsbereich, `"Controller"` der Name des Controllers und `"view"` die auszuführende Aktion mit dem Parameter `"14"`. Das *Composite-Pattern* wäre eine Möglichkeit dieses Routingverfahren zu realisieren. Das System soll:

- jeder Anfrage eine Route zuordnen können,
- um eigene Routen erweiterbar sein und
- nur in den APP-Bereich Routing zulassen.

Authentifikation und Autorisation

Um in der Anwendung spezielles, benutzerbasierendes Verhalten verwenden zu können ist es nötig Funktionen zum An- und Abmelden von Benutzern sowie zum Prüfen von Benutzerrechten zu implementieren.

Das System soll:

- Funktionen zum Abbilden von Rechten auf *Controller* und Aktionen bieten,
- das Ein- und Ausloggen von Benutzern ermöglichen und
- *Assertions* bereitstellen.

Datenbankzugriff

Um der Anwendung das dauerhafte Ablegen von Daten zu ermöglichen, ist es sinnvoll Datenbankschnittstellen bereitzuhalten. Wenn diese einheitlich gestaltet sind, erleichtert dies desweiteren auch den Austausch der verwendeten Datenbank und bietet eine zusätzliche Abstraktionsschicht. Im Idealfall können Daten so ohne *SQL-Anweisungen* zu formulieren in Datenbanken abgelegt werden.

Das System soll:

- einheitlichen Zugriff mittels *CRUD/EXEC*-Schnittstelle ermöglichen und
- ein standardisiertes generisches Interface für alle Datenbanken anbieten.

Sitzungen

Um Daten anfragenübergreifend zu speichern ist es notwendig eine Sitzungsverwaltung zu implementieren. Über sogenannte *Cookies* werden die Nutzer wiedererkannt und jedem Cookie wird eine Datei mit Nutzerdaten zugeordnet. PHP stellt hier mannigfaltige Funktionen bereit, welche in dem Sitzungsobjekt gekapselt werden.

- Das System soll eine Speicherung von Daten über einen einzelnen *HTTP*-Request hinaus ermöglichen.

Allgemein

Die hier definierten Anforderungen gelten systemweit und ergeben sich durch den Einsatz der spezifischen *Patterns* oder als logische Konsequenz aus den anderen Anforderungen.

Das System soll:

- die Separierung von Datenhaltung, Datenansicht und Kontrollfluss ermöglichen,
- die Trennung der Fachlogik vom Controller durch Fachklassen ermöglichen,
- eine Kapselung der einzelnen Anfragen in einen isolierten Request ermöglichen, welcher ein definiertes Response liefert,
- HTTP-Anfragen und -Antworten kapseln und
- Schnittstellen zur Implementierung und Anbindung von Komponenten anbieten.

Ansicht

Ansichten bereiten Daten grafisch auf und beugen mit *Templating* dem Vermischen von Logik mit Formatierungsanweisungen vor.

Ansichten sollen:

- Daten in spezieller, festgelegter Formatierung ausgeben und
- Formatierungshelfer aufrufen sowie *Templating* nutzen können.

Fachklasse

Fachklassen bereinigen den *Controller* von anwendungsspezifischer Logik. Dies dient der Steigerung der Übersichtlichkeit.

- Die Fachklassen sollen die applikationsspezifische Datenhaltung ermöglichen.

Controller

Die *Controller* koordinieren die Datenmodelle und die Datenansichten miteinander.

- Die *Controller* sollen die applikationsspezifische Logik, die zur Bereitstellung der Ansichten und dem Verwalten der Fachklassen notwendig ist, beinhalten.

Caching

Caching kann eingesetzt werden, um das Ladeverhalten der Anwendung positiv zu beeinflussen.

- Durch Caching können häufig genutzte Ressourcen zwischengespeichert werden.

3.2 Nicht-Funktionale Anforderungen

Technische Anforderungen: Das System muss auf einer LAMPP-Plattform lauffähig sein.

Wartbarkeit: Das System soll durch Zentralisierung und Strukturierung eine einfache Wartung ermöglichen.

Sicherheit: Das System soll durch entsprechende Mechanismen nicht gewünschte Anwendungsfälle ausgrenzen können.

Skalierbarkeit: Auch unter schweren Lasten soll das Systemverhalten vorhersagbar bleiben.

Erweiterbarkeit: Das System muss so entwickelt werden, dass es erweitert / modifiziert werden kann, falls die Anforderungen es verlangen.

Robustheit (Fehlertoleranz): Das System soll undefinierte Zustände verhindern und das Abfangen fehlerhafter Nutzereingaben gewährleisten.

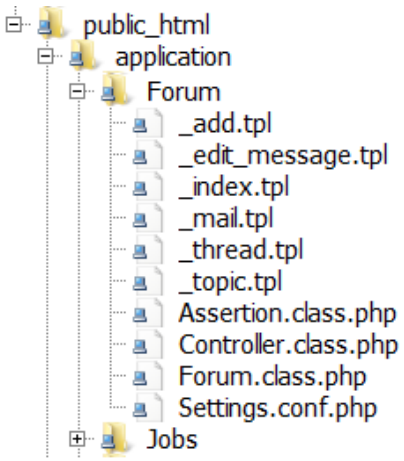
4 Entwurf eines WAF

Im folgenden Kapitel wird näher auf die Umsetzung der einzelnen Anforderungen eingegangen. Dabei wird zunächst auf die Verzeichnisstruktur eingegangen. Anschließend findet eine kurze Erläuterung der Systemkomponenten statt.

4.1 Verzeichnisstruktur

Das Framework ist sowohl strukturell als auch im Dateisystem in folgende 4 Bereiche unterteilt:

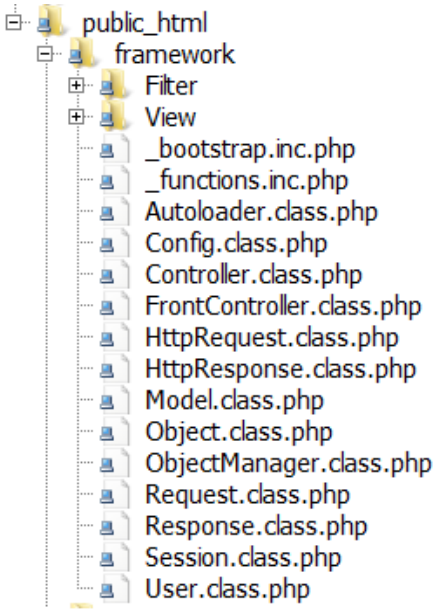
- **APP (application) - applikationsspezifischer Bereich:**



Hier sind die Fachklassen, *Controller, Views*, Konfigurationen, Anwendungsdaten sowie CSS- und JS-Dateien abgelegt. Das Verzeichnis ist von außen ansprechbar und enthält die eigentliche Anwendung.

Abbildung 10: Ordnerstruktur einer Anwendung

- **FW (framework) - frameworkspezifischer Bereich:**



Hier finden sich alle Klassen, die zur Ausführung des Frameworks notwendig sind.

Abbildung 11: Ordnerstruktur des Frameworks

- **MOD (modules) - modulspezifischer Bereich:** Als Module werden Teile wiederverwendbarer Software verstanden, die in Anwendungen eingebettet werden oder häufige Funktionen bereitstellen. Module sind wie Anwendungen aufgebaut, jedoch von außen nicht direkt ansprechbar.
- **LIB (library) - bibliotheksspezifischer Bereich:** Hier lagern Klassen, die häufig verwendete Funktionen bereitstellen.

Im Dateisystem sind desweiteren noch folgende Ordner enthalten:

- config: Enthält die Startkonfiguration des Frameworks sowie Datenbank-Zugänge und Routen.
- tmp: Kann zum Lagern von Sitzungsdaten, Dateien sowie als Cache benutzt werden.

4.2 Systemkomponenten

Das oben definierte System setzt sich aus folgenden Einzelkomponenten zusammen:

Autoloader

Der Autoloader setzt Klassennamen zu Pfaden im Dateisystem um. Über Prefixe können weitere Arbeitsbereiche definiert werden. Existiert eine Datei nicht, wird eine leere Klasse zur Laufzeit mit dem entsprechenden Klassennamen definiert. An folgendem Beispiel wird die Übersetzung des Pfades zum Klassennamen deutlich:

```
FW_Config wird zu ./framework/Config.class.php
LIB_View_HTML wird zu ./library/View/HTML.class.php
```

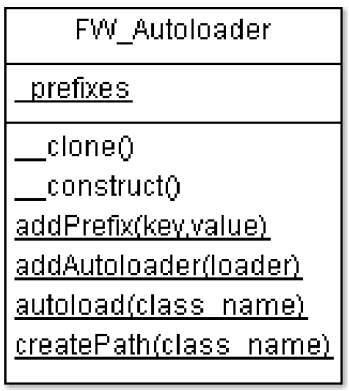


Abbildung 12: UML-Diagramm des Autoloaders

Konfiguration

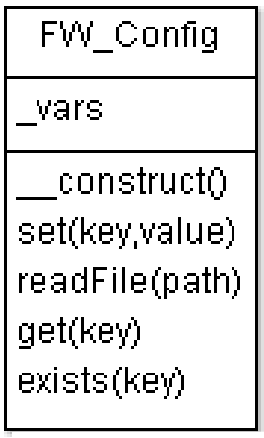


Abbildung 13: UML-Diagramm der Konfigurationsklasse

Konfigurationen werden zentral als *Arrays* in Dateien abgespeichert. Sie können von der Konfigurationsklasse eingelesen werden und stehen zur Programmlaufzeit zur Verfügung.

```
return array('FW_Model' => array(
     'adapter' => 'LIB_Model_MySQLi',
     'host'    => '<db_host>',
     'dbname'  => '<db_name>',
     'user'    => '<db_user>',
     'pwd'     => '<db_pass>',
     'prefix'  => "<prefix>"));
```

Objekt-Manager

Der Objekt-Manager kümmert sich um das Erschaffen von Objekten und dem Zuweisen von den entsprechenden Klassenkonfigurationen. Er verwaltet desweiteren die Instanzen von Singletons und dient als globaler Speicherort für Objekte. Er vereint somit die Entwurfsmuster einer *Registry* sowie eines *Builders*.

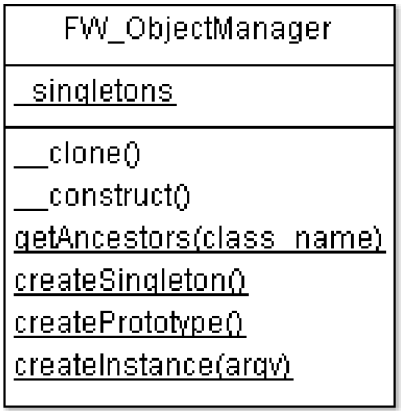


Abbildung 14: UML-Diagramm des Objekt-Managers

FrontController

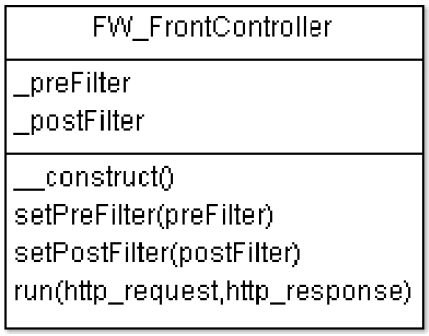


Abbildung 15: UML-Diagramm des FrontControllers

Der *FrontController* ist der zentrale Einstiegspunkt in die *HMVC*-Umgebung. Er nimmt *HTTP*-Requests entgegen, erstellt die Ursprungs-*HMVC*-Anfrage und gibt dieser die *HTTP*-Request Parameter weiter. Desweiteren führt er die *HMVC*-Anfrage aus, filtert alle Ein- und Ausgaben und gibt die entstandenen Ergebnisse an den Client weiter.

Request

Der Request kapselt einen *HMVC*-Aufruf, kümmert sich um das Routing und das Ausführen des entsprechenden Controllers. Er ist auch für die Prüfung der Benutzerrechte zuständig.

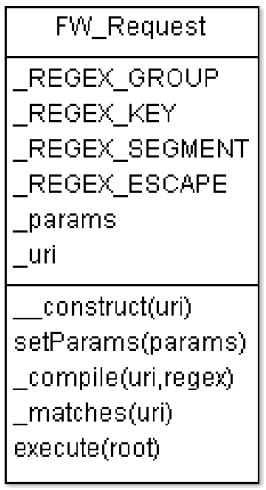


Abbildung 16: UML-Diagramm des HMVC-Requests

Routing ist der Mechanismus, der die Verbindung zwischen den *URLs* und den *Controllern* und Aktionen herstellt. Die Routen werden in der Konfigurationsdatei für den Request definiert. Es können Platzhalter, default-Werte und optionale Parameter übergeben werden. Es wird eine Liste aller Routen durchlaufen, bis eine *URL* auf ein Routing-Muster passt. Anschließend wird der dort hinterlegte *Controller* mit der entsprechenden *Aktion* aufgerufen.

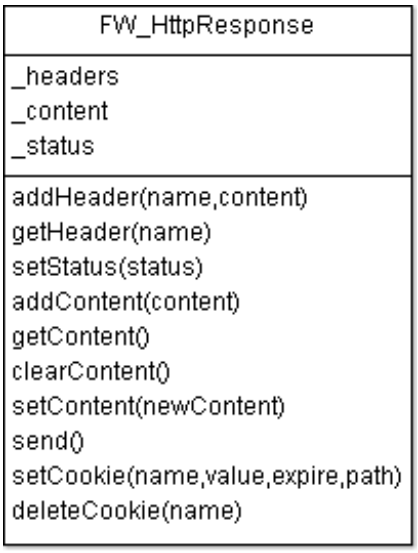
Folgende Routendefinition ist gegeben:

```
'default'  => array('uri' =>
'(<prefix>(/<module>(/<controller>(/<action>(/<params>)))))', 'defaults'
=> array('prefix' => 'app', 'module' => 'cms', 'controller' =>
'controller', 'action' => 'index'), 'regex' => array('params' => '.*',
'module' => '(error|template)'))
```

Diese lässt sich auflösen zu:

```
'<Name der Route>'  => array('uri' => '<URL-Pattern zum Vergleich>', 'de-
faults'  => array('<Default-Werte für Parameter>'), 'regex' => ar-
ray('<Parameter-Pattern zum Vergleich>'))
```

HTTP-Request



Kapselt eine *HTTP*-Anfrage in einem Objekt. Alle Anfragedaten stehen somit in diesem Objekt bereit. Dies ermöglicht einen einheitlichen Zugriff auf Anfragedaten.

Abbildung 17: UML-Diagramm des HTTP-Responses

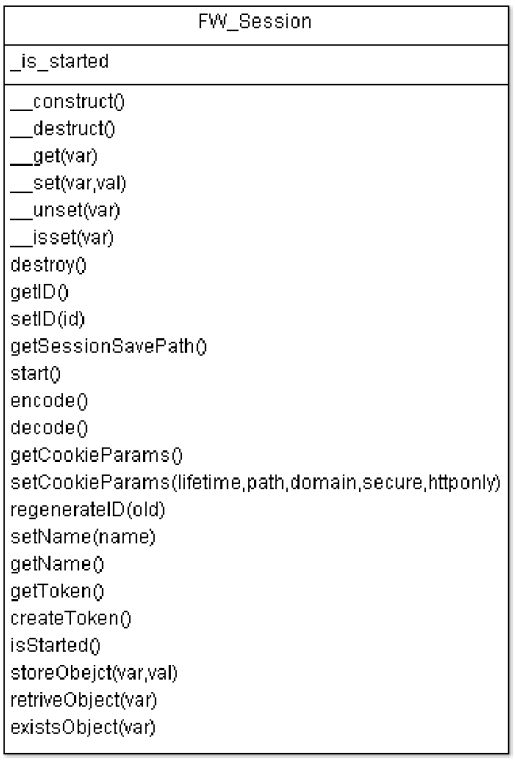
HTTP-Response

Kapselt das Ergebnis einer *HTTP*-Anfrage in einem Objekt. Alle Ergebnisdaten stehen somit in diesem Objekt bereit. Dies ermöglicht einen einheitlichen Zugriff auf die Ergebnisdaten.

Abbildung 18: UML-Diagramm des HTTP-Requests

Sessions

Sitzungen ermöglichen die *HTTP*-Request-übergreifende Speicherung von Daten. Der Server kann den Client aufgrund eines individuellen Datenfeldes im Kopf einer *HTTP*-Anfrage wiedererkennen und bereits abgelegte Daten wieder zuordnen. Dies ermöglicht interaktive Systeme mit verschiedenen Benutzern und benutzerspezifischen Ansichten. Die Session-Klasse nutzt die sogenannten "*magischen Methoden*" von PHP5, um eine einheitliche Datenschnittstelle anzubieten. Beispiel für "*magischen Methoden*" von PHP5:




```
public function __get($var) {
  $this->start();
  if(isset($_SESSION['_data'][$var])) {
    return $_SESSION['_data'][$var];
  } else {
    return null;
  }
}
public function __set($var, $val) {
  $this->start();
  return ($_SESSION['_data'][$var] =
$val);
}
public function __unset($var) {
  $this->start();
  if(isset($_SESSION['_data'][$var])) {
    unset($_SESSION['_data'][$var]);
  }
}
public function __isset($var) {
   $this->start();
   return
isset($_SESSION['_data'][$var]);
}
```

Abbildung 19: UML-Diagramm der Session-Klasse

Model

Das Model verwaltet eine Datenbankverbindung. Jede von PHP unterstützte *SQL*-Datenbank kann angesprochen werden. Gegebenenfalls müssen Anpassungen an dem Modell gemacht werden. Das Model kann wahlweise mit einem *Adapter-Entwurfsmuster* umgesetzt werden.

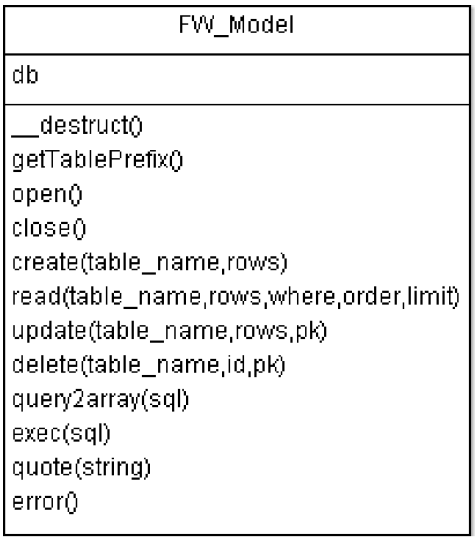


Abbildung 20: UML-Diagramm des Datenmodels

Das Model bedient sich dem weitverbreiteten *CRUD*-Paradigma, wobei *CRUD* für (C)reate, (R)ead, (U)padate und (D)elete steht. Als Datentypen werden PHP-Arrays übergeben, wobei die Array-Feldbezeichner mit den Datenbankfeldbezeichnern übereinstimmen müssen. Sollten Abfragen nicht via *CRUD* abgewickelt werden können, hat man mit den exec()- und query2array()-Funktionen die Möglichkeit, direkt auf die Datenbank zuzugreifen.

Dabei sollten die Eingabewerte mittels der quote()-Funktionen behandelt werden, um böswillige Benutzereingaben zu verhindern.

User

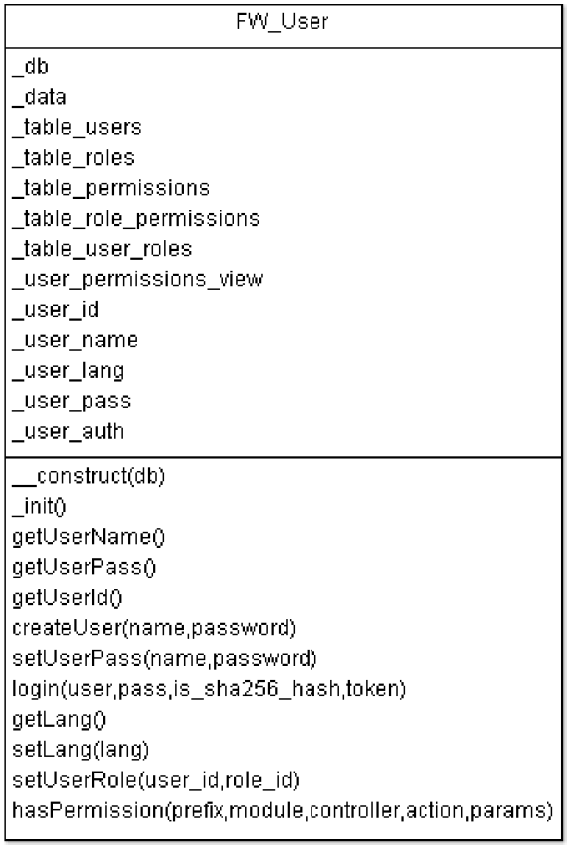
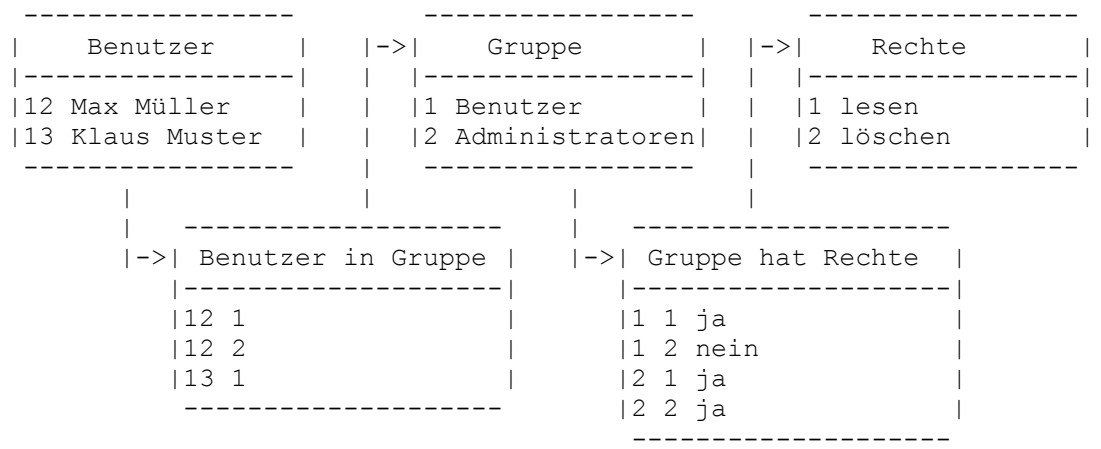


Abbildung 21: UML-Diagramm der User-Klasse

Kapselt Methoden zum Verwalten von Benutzern. Dies beinhaltet unter anderem das Authentifizieren und Autorisieren von Benutzern. Die Autorisation erfolgt über eine in einer Datenbank abgelegte Access-Control-List (*ACL*). Diese ordnet Benutzern spezielle Gruppen zu, welche wiederum mit Rechten ausgestattet werden. Somit kann jede existierende Benutzergruppe mit individuellen Rechten versehen werden. Einzelne Rechte sind definiert als Modul/*Controller/Action*-Tupel, einer Erlaubnis oder einem Verbot sowie der Benutzerzuordnung. Ein Beispiel für eine solche Zuordnung könnte sein:

[10]



Benutzer haben Lese-, aber keine Lösch-Rechte. Administratoren hingegen dürfen lesen und löschen. Max Müller ist in der Gruppe Benutzer und Administratoren. Somit darf er lesen und löschen. Klaus Muster ist jedoch nur Benutzer und darf daher nur lesen. Das Löschen ist ihm untersagt. Mittels der login()-Methode können Benutzer mit Nutzernamen und Passwort an dem instanziierten Objekt angemeldet werden. Zur Abfrage der Berechtigungen steht die hasPermissions()-Methode zur Verfügung.

4.3 Struktur

Die hier dargestellte Struktur liefert einen groben Überblick über den Aufbau des Frameworks. Bei genaueren Hinsehen werden weitere feingranulare Strukturen sichtbar. So befindet sich zum Beispiel im *FrontController* ein *Composite-Pattern* bei den Filtern, im Objekt-Manager findet sich ein *Builder-Pattern* und eine *Registry*. Die Anwendungsentwicklung findet innerhalb des grau umrandeten Blockes statt.

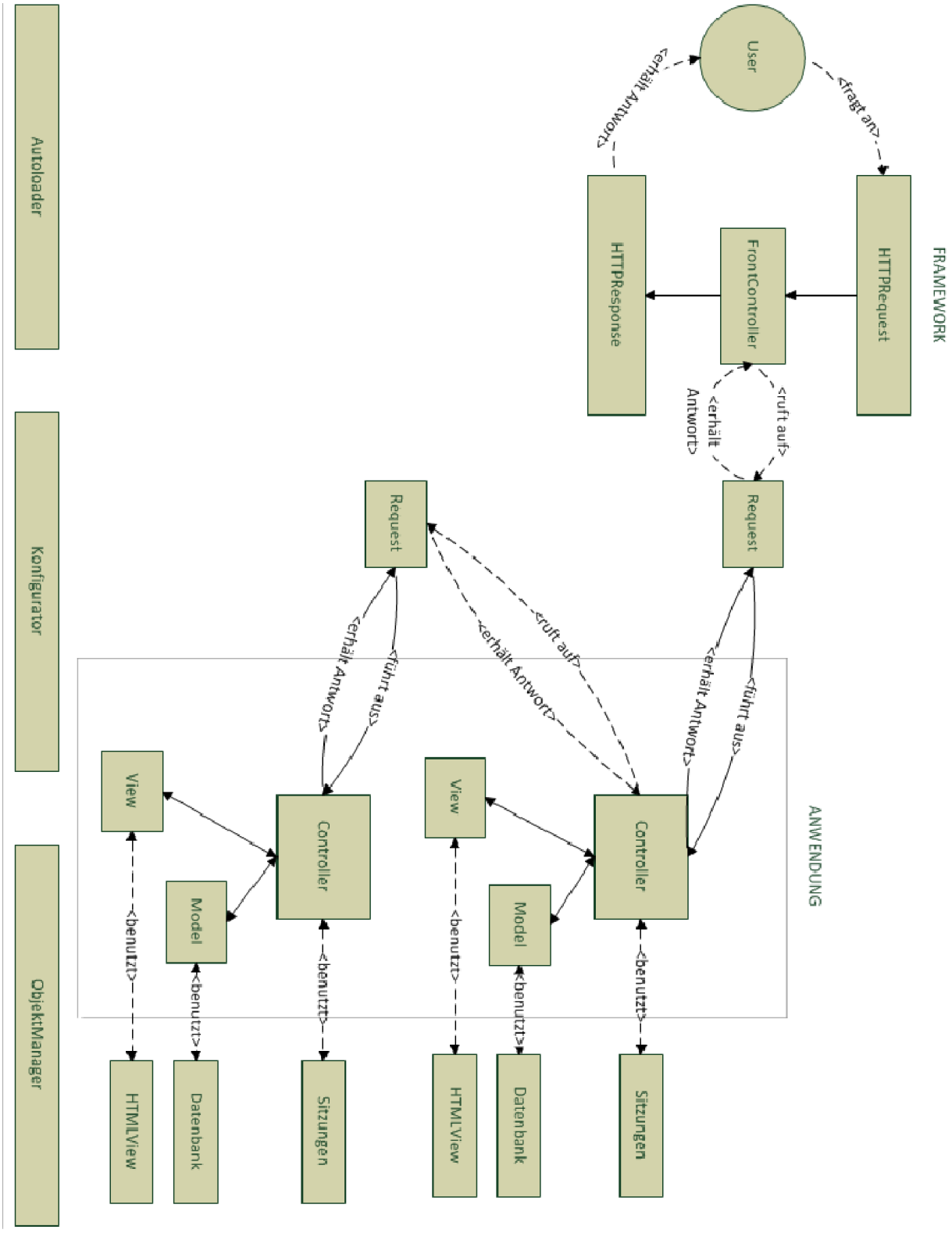


Abbildung 22: Strukturübersicht des Frameworks

4.4 Anwendungsablauf

Im folgenden wird der Ablauf der Anwendung beschrieben:

- Der Einstiegspunkt für die Anwendung ist in der index.php.
  - Diese bindet die _bootstrap.inc.php ein.
  - Anschließend wird der *FrontController* ausgeführt.
- In der _bootstrap.inc.php werden:
  - Konstanten gesetzt.
  - Globale Funktionen geladen (_functions.inc.php).
  - Autoloader initialisiert.
  - Konfigurationsobjekt initialisiert.
  - *HTTP*-Request und *HTTP*-Response initialisiert
  - *FrontController* initialisiert.
- Anschließend findet die Ausführung des FrontControllers statt:
  - preFilter werden ausgeführt.
  - Ein *HMVC*-Request wird erstellt, die *HTTP*-Parameter werden übergeben.
  - Der erstellte *HMVC*-Request wird ausgeführt, das Ergebnis anschließend dem *HTTP*-Response übergeben.
  - postFilter werden ausgeführt.
  - Der *HTTP*-Response wird zum Client übertragen.
- Ausführung des *HMVC*-Requests:
  - Die Routen werden kompiliert.
  - Anschließend werden die richtigen *Controller/Action*-Paare zur URL zugeordnet.
  - Prüfen, ob der angemeldete Benutzer die nötigen Rechte hat, den *Controller* auszuführen.
  - Einbinden der benötigten Modulkonfigurationsdatei.
  - Erstellen des *Controllers* durch den ObjektManager.
  - Ausführen der before()-Methode des Controllers.
  - Ausführen der Action des *Controllers*.
  - Ausführen der after()-Methode des Controllers.
  - Zuletzt wird ein Response-Objektes mit Ergebnis zurückgegeben.

Innerhalb eines Controllers können weitere *HMVC*-Subrequests gestartet werden. Sie durchlaufen alle Punkte von "Ausführung des *HMVC*-Requests".

5 Umsetzung

Im folgenden Kapitel wird anhand der Beispielapplikation auf die Funktionsweise des Web-Application-Frameworks eingegangen. Dabei wird zunächst der Vorgang der Anbindung des Frameworks erläutert. Anschließend wird auf Basis der hier vorgestellten Beispielanwendung die Funktionsweise dargestellt.

5.1 Anbindung

Die zu erstellende Webanwendung muss die Datei "_bootstrap.inc.php" inkludieren. Anschließend müssen benutzerdefinierte Routen erstellt und in den entsprechenden Konfigurationsdateien gesichert werden. Um nur einen Einstiegspunkt zu erlauben werden entsprechende ".htaccess"-Direktiven angelegt. Der Datenbank-Zugang wird in der entsprechenden Konfigurationsdatei hinterlegt.

Um neue Anwendungen zu entwickeln, wird im "APP"-Verzeichnis ein neuer Ordner angelegt. Dieser enthält die Controller, die Templates, anwendungsspezifische Daten, eine Konfigurationsdatei sowie die Fachklassen. Um größtmögliche Konsistenz zu gewährleisten, sollten die vom Framework bereitgestellten Methoden und Objekte zur Datenhaltung verwendet werden.

Ab hier können sich die Wege zur Entwicklung der eigentlichen Anwendung stark unterscheiden.

5.2 Beispielanwendung

Bei der hier vorgestellten Anwendung handelt es sich um ein Forum mit den hierfür typischen Anwendungsfällen. Das hier übergebene Projekt enthält darüber hinaus jedoch noch weitere Anwendungen. Da der Anwendungsablauf jedoch standardisiert ist, haben die folgenden Erklärungen beispielübergreifende Gültigkeit.

Da das Routing der ankommenden Anfragen in dem *HMVC*-Request-Objekt stattfindet, müssen als erstes die notwendigen Routen in der Konfigurationsdatei angelegt werden, um die betreffenden *URLs* einem *Controller* zuordnen zu können. Die Daten einer *HTTP*-Anfrage werden vom *FrontController* dem *HMVC*-Request-Objekt übergeben. Dieses führt das Routing anhand der in der Konfigurationsdatei hinterlegten Routen durch, erstellt den *Controller* und prüft die Benutzerrechte.

Damit ein Zugriff auf den entsprechenden *Controller* möglich ist, müssen die entsprechenden Nutzer, Rollen und Rechte in den Datenbank-Tabellen für die Zugriffsverwaltung angelegt werden.

Anschließend muss im APP-Verzeichnis die neue Anwendung angelegt werden. Dabei hat der anzulegende Ordner den Anwendungsnamen zu tragen. Dieser Name muss mit dem Eintrag für das Routing und den Einträgen in der Zugriffsliste übereinstimmen.

Im neu erstellten Ordner kann jetzt der *Controller* erstellt werden. Dieser muss von FW_Controller erben und wenigstens die index_action() implementieren. Da sich die hier beschriebene Anwendung einem *Two-Step-View-Pattern* [Fowler2009, S.402] bedient, erbt der Anwendungs-*Controller* nicht direkt von FW_Controller, sondern vom Seitentemplate. Dies ist in diesem Fall MOD_Template_Site, welches Funktionen zum vollständigen Rendern der Seite beinhaltet. Der Anwendungs-*Controller* beinhaltet nur die eigentliche Anwendungslogik, verwaltet das genutzte Datenmodell und die Ansichten. Er kann über eine Konfigurationsdatei mit zusätzlichen applikationsspezifischen Daten versehen werden, z.B. um Mehrsprachigkeit oder Zugangsdaten bereitzustellen.

Die Anwendung wird jetzt in ihre einzelnen Bestandteile zergliedert. Ansichten werden dabei in Templates ausgelagert, die Anwendungslogik in Fachklassen. Der *Controller* arbeitet hierbei als *Vermittler* zwischen den einzelnen Instanzen. Jede Aktion, die ein Benutzer durchführen kann, wird dabei auf eine Funktion im *Controller* abgebildet. Typischerweise treten die bereits oben benannten *CRUD*-Operationen auf den anwendungsspezifischen Datenstamm angewandt auf.

So können in einem Forum alle Themengebiete (so genannte Topics) angezeigt werden. Nutzer mit den entsprechenden Rechten können Themen hinzufügen, bearbeiten und löschen (*CRUD*). Innerhalt dieser Themengebiete werden einzelne Fragestellungen diskutiert (so genannte Threads). Hier können alle am System angemeldeten Benutzer Fragestellungen hinzufügen, bearbeiten und löschen (*CRUD*). Innerhalb dieser einzelnen Fragestellungen können Benutzer Antworten verfassen sowie bearbeiten (*CRUD*).

Der folgende *Controller* enthält die oben beschriebene Funktionalität.

Abbildung 23: Anwendungscontroller des Forums

Fragt ein Nutzer mittels Eingabe einer *URL* in den Browser die Anwendung an, empfängt der *FrontController* die Anfrage, filtert bei Bedarf die Eingaben, erstellt eine Anfrage und führt diese aus. Die erstellte Anfrage prüft die Routen und Benutzerrechte und lädt anschließend den geeigneten *Controller*. Jetzt wird die entsprechende Aktion im *Controller* ausgeführt und der Rückgabewert weitergereicht bis zum *FrontController*. Dieser liefert das Ergebnis der Anfrage an den Nutzer aus. Jede in Abbildung 19 gezeigte Methode ist eine solche Aktion. Aktionen vermitteln direkt zwischen der Datenansicht und dem Datenmodell und steuern die Verarbeitung. Im Ordner der Anwendung finden sich Template-Dateien, welche ähnliche Bezeichner wie die verfügbaren Aktionen haben. Die Anwendungslogik wurde in die nach der Anwendung benannten Fachklasse ausgelagert. Die Fachklasse verwaltet desweiteren die Datenbankanbindung.

6 Zusammenfassung

Das erstellte Framework erfüllt alle in der Anforderungsanalyse definierten Bedingungen. Im Rahmen der Arbeit wurde jedoch wegen Zeit und Umfang auf einige Features verzichtet. Dies beinhaltet unter anderem eigene Fehlerbehandlung und komfortablere Debugging-Möglichkeiten. PHP5 umfasst hier jedoch für den vorgestellten Einsatzbereich ausreichende Boardmittel. Das Framework arbeitet sehr ressourcenschonend und erlaubt komplexe Anwendungen mit wenig CPU- und Speicherverbrauch. Caching-Mechanismen sind momentan nicht Bestandteil des Frameworks, können aber problemlos durch Anpassen der Datenbank- oder Ansichtsklasse nachgerüstet werden. Die hier vorgestellte Beispielapplikation zeigt die Praxistauglichkeit und steckt den Einsatzbereich des Frameworks ab. Die puristische Reduzierung des Frameworks auf so viel Abstraktion wie nötig, jedoch nicht wie möglich, kann als Streitthema betrachtet werden. So wäre es oft möglich den Sachverhalt weiter zu zergliedern, Namespaces zu nutzen, mehr Interfaces einzusetzen und die eingesetzten Patterns feiner zu strukturieren. Ist dies gewünscht, gibt es gute andere Frameworks die diesen Komfort bieten, z.B. Kohana und FLOW3. Das hier vorgestellte Produkt konzentriert sich zur Vereinfachung nur auf die notwendigen Eigenschaften eines Frameworks und gibt einen Einblick in die Funktionsweise von Web-Application-Frameworks.

Glossar

abstrakte Fabrik	Entwurfsmuster aus der Klasse der Erzeugermuster, dient dem Erschaffen von Objekten
ACL	Access-Control-List, Zugriffsliste zum Speichern von Benutzerrechten
Array	Datenfeld zur Ablage von mehreren Variablen
Assertion	Zusicherung, die erfüllt sein muss, um eine Operation auszuführen
Autoloader	Komponente, die benötigte Dateien zur Laufzeit läd
Controller	Vermittler und Steuerungsschicht zwischen dem Datenmodell und der Datenansicht
CRUD-Paradigma	Create-Read-Update-Delete, typische Operationen, die auf Datensätze angewendet werden
DI	Dependency Injection, methodische Herangehensweise um Abhängigkeiten zu zentralisieren
FrontController	Einstiegspunkt einer Applikation
HMVC/PAC	Hirachical Model-View-Controller/Presentation Abstraction Control, Strukturmuster um Anwendungen in Komponenten zu zerlegen
HTTP	HyperText Transfer Protocol, Protokoll zum Datenaustausch in Computernetzen
IoC	Inversion of Control, Paradigma zum Anwendungsablauf, Steuerung nach dem sog. Hollywoodprinzip: "Don't call us - we call you" [GoF2011, S.369]
Konstruktor	Methode, die sofort nach der Erschaffung des Objektes ausgeführt wird

loose coupling	Lose Kopplung, Abhängigkeitsbeziehung zwischen Objekten, Zustand, der das Austauschen von Komponenten ermöglicht, ohne Abhängigkeitsprobleme zu bekommen
magische Methoden	Methoden mit speziellen Fähigkeiten
Middleware	Zwischenanwendung, welche die Aufgabe hat der darüber liegenden Anwendung Schnittstellen zur Verfügung zu stellen
Model	Datenmodell um Benutzerdaten zu Verarbeiten und aufzubewahren
Namespace	Namensbereich, in dem ein Name Gültigkeit hat
ORM	Objektrelationales Mapping, Verfahren um auf Daten in einer relationalen Datenbank objektorientiert zuzugreifen
OSI-Model	Open-Systems-Interconnection-Model, Modell, welches die Datenübertragung zwischen Computern beschreibt
Prototyping	Agile Entwurfsmethode um dem Kunden schnellstmöglich ein lauffähiges Programm zu zeigen
Registry	Strukturmuster um Objekte/Konfigurationen zentral zu lagern
Routing	Vorgang um den passenden *Controller* für eine Anfrage zu finden
Session	Nutzersitzung, ermöglicht zu einem Nutzer anfragenübergreifende Informationen aufzubewahren
Singleton	Strukturmuster um maximal eine Instanz von einem Objekt zu bekommen
Template	Schablone um eine Datenansicht zu erstellen

Two Step View	Strukturmuster, bei dem eine Datenansicht aus zwei Komponenten besteht
URL	Uniform Resource Locator, eindeutige Zeichenfolge um den Ort von Dokumenten im Internet zu beschreiben
WAF	*Web-Application-Framework*, Halbzeug welches Lösungsansätze für typische Probleme bei der Anwendungsentwicklung im Webbereich anbietet

Literatur

[Fowler2009] Fowler, Martin: Patterns für Enterprise Application-Architekturen, Heidelberg, mitp, 2009.

[GoF2011] Gamma,Helm, Johnson, Vlissides: Entwurfsmuster, München, Addison-Wesley, 2009.

[Oestereich2009] Oestereich, Bernd: Analyse und Design mit UML 2.3, München, Oldenbourg, 2009.

[1] https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol, verfügbar am 30.06.2014.

[2] https://de.wikipedia.org/wiki/Framework, verfügbar am 30.06.2014.

[3] https://de.wikipedia.org/wiki/Entwurfsmuster, verfügbar am 30.06.2014.

[4] https://de.wikipedia.org/wiki/Prototyping_(Softwareentwicklung), verfügbar am 30.06.2014.

[5] https://en.wikipedia.org/wiki/Coding_conventions, verfügbar am 30.06.2014.

[6] https://de.wikipedia.org/wiki/Inversion_of_Control, verfügbar am 30.06.2014.

[7] https://de.wikipedia.org/wiki/Dependency_Injection, verfügbar am 30.06.2014.

http://martinfowler.com/articles/injection.html, verfügbar am 30.06.2014.

[8] https://de.wikipedia.org/wiki/Container(Entwurfsmuster), verfügbar am 30.06.2014.

[9] http://www.javaworld.com/article/2076128/design-patterns/hmvc--the-layered-pattern-for-developing-strong-client-tiers.html, verfügbar am 30.06.2014.

https://en.wikipedia.org/wiki/Hierarchical_model-view-controller, verfügbar am 30.06.2014.

[10] https://de.wikipedia.org/wiki/Framework, verfügbar am 30.06.2014.

[11] https://de.wikipedia.org/wiki/OSI-Modell, verfügbar am 30.06.2014.

Quellen der Abbildungen

Abbildung 1:

http://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_HowTo.html

Abbildung 2:

http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Vorgehensmodell/Prototyping/index.html

Abbildung 3:

http://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/

Abbildung 5:

http://code.tutsplus.com/tutorials/hmvc-an-introduction-and-application--net-11850

Abbildung 6:

https://de.wikipedia.org/wiki/Datei:Front_Controller.svg

Abbildung 7:

https://de.wikipedia.org/w/index.php?title=Datei:Erbauer-Entwurfsmuster.jpg

Abbildung 8:

https://de.wikipedia.org/wiki/Datei:Singleton.svg

Abbildung 9:

https://de.wikipedia.org/w/index.php?title=Datei:Kompositum_Objekte.png

Anlagen

Alle Anlagen befinden sich auf der beigefügten CD.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Radebeul, den 30.06.2014

Arne Wenzel