Seminararbeit im Rahmen des Studiengangs
Scientific Programming

Fachhochschule Aachen, Campus Jülich

Fachbereich 9 – Medizintechnik und Technomathematik

Hierarchical Model-View-Controller Entwurfsmuster in der Web-Entwicklung

Jülich, den 30. Januar 2020

Daniel Herweg

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema
Hierarchical Model-View-Controller Entwurfsmuster in der Web-Entwicklung
selbstständig verfasst und keine anderen als die angegebenen Quellen und
Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften
wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und
die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer
Studien- oder Prüfungsleistung war.
Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren
und auf Verlangen dem Prüfungsamt des Fachbereiches
Medizintechnik und Technomathematik auszuhändigen.

Name: Daniel Herweg

Aachen, den _____

Unterschrift der Studentin/des Studenten

Über die Zeit hat sich Model-View-Controller (MVC) als häufig genutztes Entwurfsmuster für die Web-Entwicklung herauskristallisiert. Die Trennung von Daten in Modell und deren Darstellung in View sowie deren Programmsteuerung in Controller bringt bereits einige Vorteile mit sich. So z.B. wird die Wiederverwendbarkeit des Models und eine leichte Erweiterung einer der drei Teile gewährleistet. Mit zunehmender Komplexität verliert ein Programmierer bei MVC allerdings schnell den Überblick. Hier bietet Hierarchical Model-View-Controller (HMVC), eine Erweiterung von MVC, eine Ablösung. Im Gegensatz zu MVC, wo bei größeren Web-Seiten ein unübersichtlicher Controller entsteht, wird bei HMVC eine Web-Anwendung auf mehrere kleinere Pakete im MVC-Stil herunter skaliert. Hier werden ausgehend von einem Controller Anfragen an Unter-Controller weitergeleitet. Diese Aufteilung in viele einzelne Module sorgt für eine gute Organisation innerhalb der Applikation. Im Gegensatz zu MVC, wo man lediglich einen ganzen Block weiterverwenden konnte, können hier einzelne MVC-Pakete sowie deren einzelne Teile wiederverwendet werden. Ebenfalls verbessert sich die Erweiterbarkeit von einzelnen Komponenten, da bei MVC oft große Teile des Controllers neu geschrieben oder kopiert werden müssen, während man bei HMVC sich aus den vielen kleinen Teilen alles bereits Vorhandene zusammenstellen kann. Ein weiterer Vorteil gegenüber MVC bildet sich dabei ab, dass für Veränderungen einer Web-Seite nun nicht mehr das gesamte View neu ausgeführt werden muss. Somit verhindert HMVC den bei MVC oft auftretenden Data-Leak. In der folgenden Arbeit wird sich genauer mit HMVC auseinandersetzt und am Beispiel einer Web-Anwendung die Vorteile von HMVC gegenüber MVC dargestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Motivation und Einleitung

Viele Dienste werden den Mitarbeitern des Forschungszentrum Jülich über das Intranet angeboten. Für Menschen außerhalb des Forschungszentrums stellt die Zentralbibliothek aber auch viele Web-Seiten und Web-Anwendungen bereit.

Diese sollen aber nicht nur schön aussehen, sondern auch langfristig verwaltet und aktualisiert werden. Dafür ist es von essenzieller Wichtigkeit, dass nicht nur der Programmierer, welcher für deren Erstellung verantwortlich ist, diese versteht, sondern auch jeder nachfolgende Programmierer sollte der Struktur und der Denkweise dahinter zurechtkommen können. Um dies umzusetzen, muss der Programmcode hinter dem Web-Projekt klar erkennbar strukturiert sein. Am besten geschieht dies mit einem Design Pattern. Den Ursprung eines Design-Pattern-Kataloges verdanken wir der Gang of Four(GOF). [1] Ein in der Zentralbibliothek häufig genutztes Entwurfs-Muster ist dabei das Modell-View-Controller-Muster (MVC). Bei diesem wird das Programm in 3 Teile eingeteilt: das Modell, die Daten und in einen Controller.
Viele Web-Projekte, insbesondere die großen, zeigen aber, wo MVC noch Schwachstellen besitzt. Besonders problematisch wird MVC bei der Skalierbarkeit von Projekten. Dadurch, dass alles an Logik hinter dem Projekt in wenige Controller fällt, können diese sehr schnell zu große Ausmaße annehmen. Das stellt insbesondere ein Problem dar, wenn das Projekt um Funktionalität erweitert werden soll, und sich der Programmierer in der Zeit gewechselt hat. Der neue Programmierer muss sich dann erst einmal in alles, was die Controller können, einarbeiten, um zu wissen, was er benutzen kann und was er neu programmieren muss. So kann es also vorkommen, dass das Projekt, trotz vorhandener Strukturierung den Anforderungen nicht mehr gerecht werden kann. Darum wählen viele Entwickler mittlerweile andere oder modernere Ansätze zur Strukturierung ihrer Web-Projekte. Reines MVC ist sozusagen out. Also brauchen wir eine Alternative zu MVC. Ohne lange zu suchen, stößt man hier schnell auf HMVC. Eine Erweiterung von MVC, welche verspricht einiges besser zu machen. Darunter z.B. eine deutlich bessere Skalierbarkeit und bessere Überschaubarkeit der Struktur. Doch in wie weit ist HMVC besser als MVC? Dies gilt es in der folgenden Arbeit herauszufinden, damit entschieden werden kann, ob und wann HMVC für Web-Projekte in der Zentralbibliothek genutzt werden kann. Dies wird unter anderem anhand einer Beispiel Web-Anwendung genauer untersucht.

[1] Design Patterns(GOF),2015

2. Grundlagen

2.1. Modell-View-Controller(MVC)

Model-View-Controller oder auch nur MVC genannt ist ein Entwurfsmuster zur Modellierung eines Programmes. Ziel dieses Musters ist es hierbei ein flexibles Programm entstehen zu lassen. Dies bedeutet, dass eine spätere Änderung oder eine Wiederverwendung einzelner Teile ermöglicht werden soll. Bei MVC wird ein Programm in ein Modell, eine View und einen Controller aufgeteilt. Die Kombination aus Modell, View und Controller wird als eine Triade betrachtet. Normalerweise arbeitet MVC mithilfe des Beobachter-Musters, um der View eine Änderung der Daten mitzuteilen.

2.1.1. Modell(Daten)

Das Modell in MVC muss 3 grundlegende Eigenschaften erfüllen:
1. Es muss einen Zustand verwalten können.
2. Es muss darstellbar sein.
3. Es muss auf Aufforderungen zu Änderungen reagieren können.
Grundlegend werden im Modell alle für ein Programm wichtigen Informationen gespeichert. Darum entspricht das Modell in vielen Anwendungen einer Datenbank.

2.1.2. View(Oberfläche)

View(Oberfläche): Die View ist für die Darstellung der Daten aus dem Modell verantwortlich. Die Darstellungsarten sind dabei nicht eingeschränkt von Textausgabe über 3D Bilder bis hin zu reiner Sprachausgabe.

2.1.3. Controller(Steuerung)

Der Controller übernimmt die Aufgabe zu verstehen was bei eingehenden Anfragen getan werden muss. Anschließend aktualisiert er die Daten im Modell.

2.1.4. Business Logik

Bei MVC ist es nicht gänzlich definiert, wo sich die Business Logik zu befinden hat. Es gibt dafür mehrere Möglichkeiten. Zunächst bietet sich uns die Möglichkeit die gesamte Logik in den Controller zu legen. Alternativ dazu könnten wir unsere Logik in das Modell verlegen. Dies wird oft angewendet, wenn sich die Logik stark auf

die gegebenen Daten bezieht. Zuletzt gibt es noch die Möglichkeit beide Wege zu kombinieren und abhängig von der Funktionalität unsere Logik in den Controller oder in das Modell zu legen.

2.1.5. Pull- versus Push-MVC

Beim klassischen MVC handelt es sich um eine Push Umsetzung von MVC. Der Controller entscheidet, welche View angezeigt wird. Anfragen von der View werden übers Observer-Pattern an den Controller mitgeteilt. Dieser aktualisiert dementsprechend die Daten auf dem Modell und gibt die richtige View wieder. Bei einer Veränderung der Daten auf dem Modell wird die View mittels des Observer-Patterns Informiert(Push), dass eine Änderung statt gefunden hat. Diese besorgt sich dann die neuen Daten und zeigt sie an.

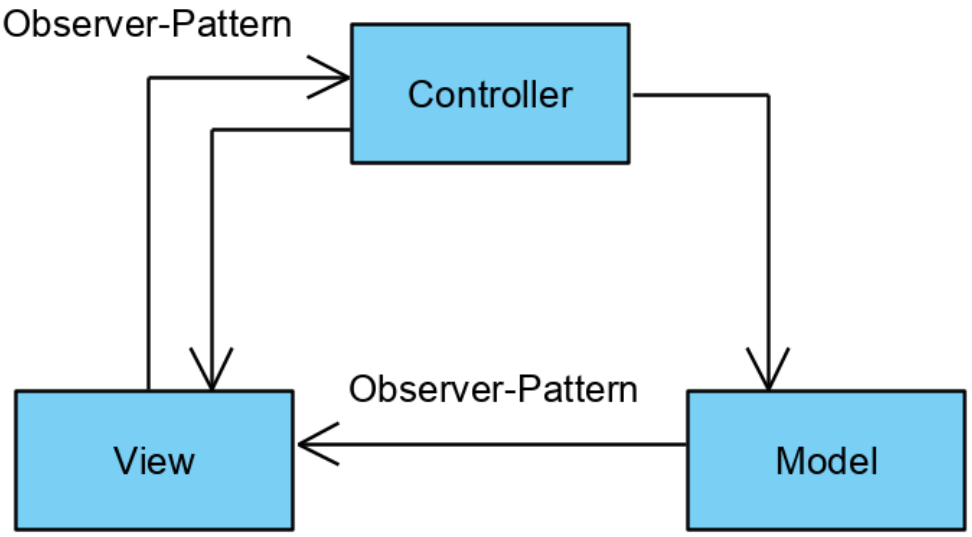

Abbildung 2.1.: Push MVC

Bei Pull-MVC setzen wir das Observer-Pattern beim Modell nicht um. Stattdessen holt sich die View, sobald sie aufgerufen wird, oder gegebenenfalls in regelmäßigen Abständen, die Daten vom Modell(Pull).

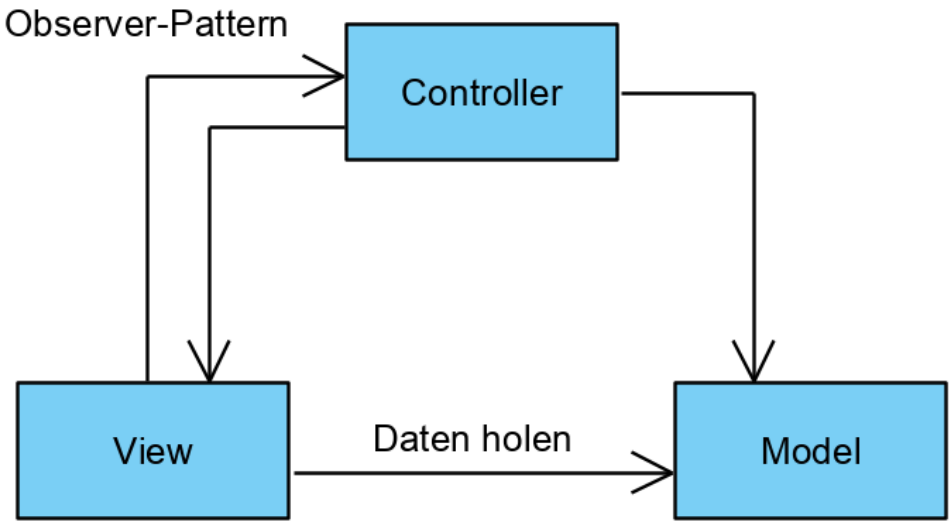

Abbildung 2.2.: Pull MVC

2.2. Web-Anwendung

Das World Wide Web ist voll mit Web-Seiten und Web-Anwendungen. Aber wo ist der Unterschied? Unterschieden wird hierbei hauptsächlich nach dem Zweck des Web-Projekts. Web-Seiten dienen hauptsächlich der Informationsbeschaffung und bestehen aus größtenteils statischen Elementen. Ein gutes Beispiel dafür sind Blogs, welche lediglich ein bisschen Text anzeigen. Web-Anwendungen hingegen dienen der Interaktion mit der Seite. So kann man sich auf diesen meist anmelden und beeinflussen, was auf der Seite angezeigt wird. Beispiele hierfür sind Einkaufsseiten wie Amazon oder Soziale-Media Seiten wie Facebook. (Bild https://byteq.com/de/b/webseite-webanwendung-unterschied)

2.3. Architekturmuster/Entwurfsmuster

Muster beschreiben im Allgemeinen, wie man für ein auftretendes Problem eine Lösung finden kann. Dies geschieht mittels einer Vorlage, welche lediglich für den Anwendungsfall angepasst werden muss. [1] Architekturmuster und Entwurfsmuster lassen sich nicht komplett voneinander trennen. Es gibt fließende Übergänge zwischen ihnen. So kann MVC sowohl als Architekturmuster sowie auch als Entwurfsmuster verwendet werden.

[1] Design Patterns(GOF),2015,S27

2.3.1. Architekturmuster

Architekturmuster (architecture pattern) beschreiben Systemstrukturen, die die Gesamtarchitektur eines Systems festlegen. Sie spezifizieren, wie Subsysteme zusammenarbeiten.

2.3.2. Entwurfsmuster

Entwurfsmuster (design pattern) geben bewährte generische Lösungen für häufig wiederkehrende Entwurfsprobleme an, die in bestimmten Situationen auftreten. Sie legen die Struktur von Subsystemen fest.

3. Vor und Nachteile von MVC bei Webanwendungen

3.1. Vorteile

- Durch die Aufteilung in drei lose gekoppelte Subsysteme wird es deutlich einfacher Teile des Programmes wiederzuverwenden. Um eine Seite der gleichen Firma zu erstellen, muss man z.B. nicht komplett von vorne anfangen, sondern kann die View oder das Modell wiederverwenden.
- Auch der Austausch von einzelnen Komponenten, sollte beispielsweise eine Anpassung der View auf verschiedenen Benutzeroberflächen vorgesehen sein, wird durch die Dreiteilung deutlich vereinfacht. So kann Beispielsweise eine Änderung des Corporate-Design einfach vorgenommen werden.
- Während der Entwicklung lässt sich die Arbeit leichter auf verschiedene Personen aufteilen, welche sich auf die einzelnen Bereiche spezialisieren.
- Durch die Aufteilung lassen sich die Komponenten einfach erweitern, ohne dass direkt das ganze Programm umgeschrieben werden muss.

3.2. Nachteile

- Wegen der komplexeren Struktur des Programmes erhöht sich auch der Arbeitsaufwand für den/die Programmierer.
- MVC kann nicht gut skalieren. Das bedeutet für immer größer werdende Anwendungen, dass für kleine Änderungen, enorme Teile des Programms angepasst oder neu geschrieben werden müssen.
- Aufbauend darauf wird es auch zunehmend schwieriger neue Mitarbeiter in ein bestehendes System einzuarbeiten, da die einzelnen Bereiche viel zu unübersichtlich werden.

4. HMVC

HMVC ist eine Erweiterung für MVC. Dabei soll HMVC in erster Linie das Problem der Skalierbarkeit von MVC lösen. Die erste Erwähnung von HMVC war im July 2000 auf der Website JavaWorld[1]. Bei HMVC sorgen wir dafür, dass die MVC-Triaden klein und überschaubar bleiben. Dafür teilen wir unsere Web-Anwendung einfach in mehrere MVC Triaden auf. Diese werden nach dem alten Prinzip von MVC aufgebaut. Diese funktionieren unabhängig von den anderen Triaden und können beliebig in anderen Programmen benutzt werden. Dadurch ist es auch unwichtig, wo innerhalb des Programmes diese Triade liegt, solange sie von überall aus erreichbar ist.

4.1. Controller

Alle Anfragen an eine Triade laufen über den Controller dieser Triade. Dort wird vom Controller aufgelöst, was zu tun ist. Entweder stellt der Controller fest, dass die Anfrage außerhalb seiner Zuständigkeitsbereiches liegt, oder er kommuniziert mit dem Model, um sich die entsprechenden Daten zu holen, und bereitet dann die View mithilfe dieser Daten vor. Diese View gibt er anschließend an den Controller, welcher ihn aufgerufen hat weiter.

4.2. View

Die View kann grundlegend gelassen werden wie bei MVC.

4.3. Modell

Die Triaden sind meist für ganze Bereiche verantwortlich. So könnte es eine eigene Triade für z.B. Bücherverwaltung geben. Diese würde sich um alles, was mit Büchern getan wird kümmern. Das Modell in dieser Triade würde sich dann mit allem beschäftigen, womit die Datenbank mit Büchern zu tun hat. Braucht eine andere Triade etwas zur Bücherverwaltung, so wird entweder das Modell der Bücherverwaltung wiederverwendet oder die Anfrage muss über den Controller der Bücherverwaltung geschehen.

[1] Jason Cai, Ranjit Kapila and Gaurav Pal: HMVC: The layered pattern for developing strong cliet tiers - Anhang Link 2

4.4. Business Logik

Da HMVC viel Wert auf Modularisierung legt empfiehlt es sich die Logik komplett in die Controller zu legen.

4.5. Funktionsweise

HMVC kann sowohl mit Push-MVC als auch mit Pull-MVC umgesetzt werden. Jede Anwendung mittels HMVC besitzt eine Triade, welche an der Spitze der Aufrufkette steht. Diese Triade erhält alle einkommenden Anfragen und leitet diese weiter. Jeder weitere Controller löst dann jeweils die Anfrage auf und macht gegebenenfalls weitere Anfragen an andere Controller. Sobald ein Controller mit seiner Funktionalität fertig ist, gibt dieser seine Ergebnisse oder Views an seinen aufrufenden Controller weiter und der oberste Controller sammelt diese und bereitet die View vor, welche dann dem Benutzer ausgegeben wird. In der Abbildung 4.1 wird gezeigt wie die Tria-

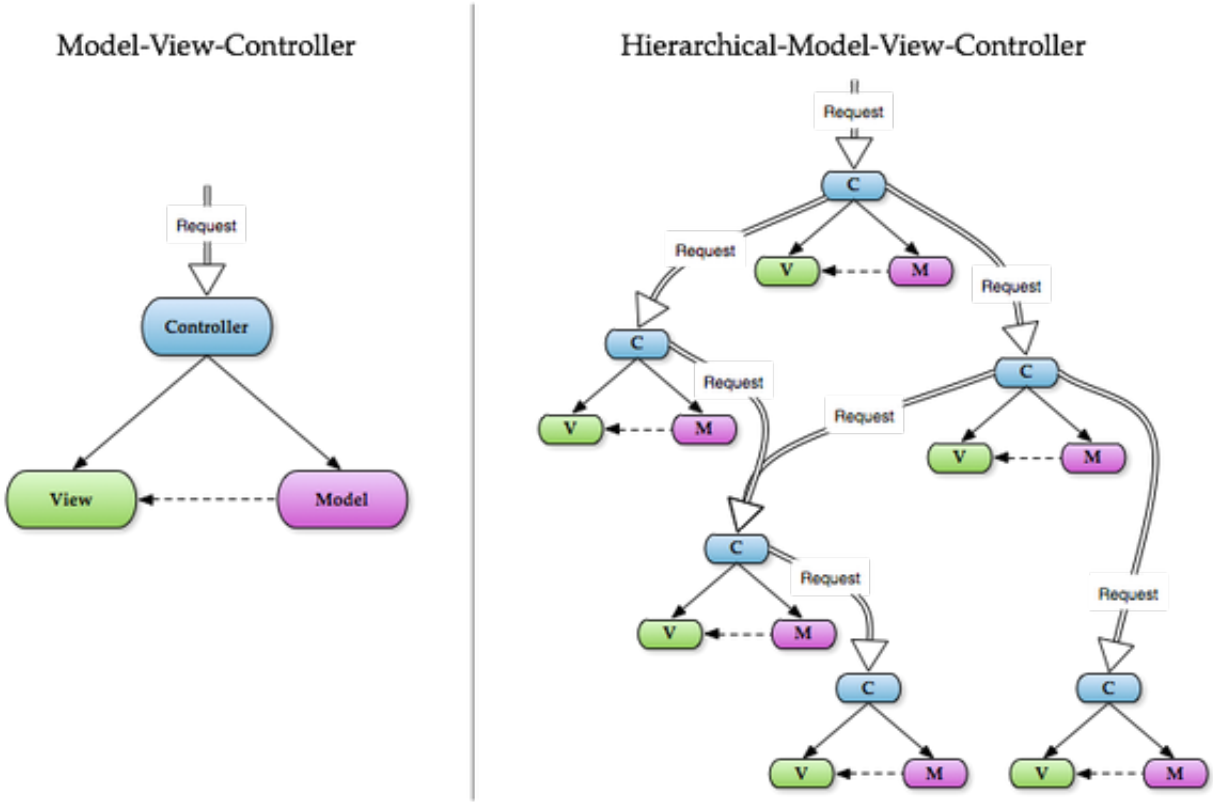

Abbildung 4.1.: HMVC-Struktur(basierend auf Push-MVC) Quelle: Anhang Link 15

den, in HMVC, zusammen das Gerüst des Programmes bilden. Die oberste Triade ist verantwortlich alle eingehenden Anfragen, an die entsprechenden Gebiete weiter

zu leiten. Alle nachfolgenden Triaden tragen dann jeweils die Verantwortung für genauere Teilgebiete. Die meisten Web-Projekte haben mittlerweile eine Registrier- und Anmeldefunktion. Diese würden Beispielsweise von einer eigenen Triade übernommen, welche alle Anfragen, die sich auf einen Benutzer beziehen, verarbeiten kann und als einziges ein Modell, mit Zugriff auf die Benutzerdaten besitzt.

5. Beispiel Web-Anwendung

5.1. Beschreibung

Ziel dieses Kapitels ist eine Web-Anwendung, anhand welcher der Unterschied zwischen einer Web-Anwendung mit MVC und einer Web-Anwendung mit HMVC verdeutlicht werden kann, zu zeigen. Dafür nehmen wir uns vor, eine ganz klassische Web-Anwendung zu entwickeln - ein Forum.
Dafür wurde, bei den MVC Triaden, ein Ansatz gewählt, bei welchem die gesamte Logik in die Controller fällt.

Das Forum soll über folgende Funktionalität verfügen:

- Ein Benutzer soll sich an- und abmelden sowie sich registrieren können.
- Es soll jedem Benutzer möglich sein Themen und Beiträge zu diesen Themen anzusehen.
- Jeder Benutzer kann die allgemeinen Informationen zur Seite einschauen.
- Eingeloggte Benutzer sollen außerdem die Möglichkeit haben:
  - Neue Themen und Beiträge zu erstellen.
  - Sofern sie Besitzer eines Themas sind, diesem Tags zu geben.
  - Alle von ihnen erstellte Themen, Beiträge und Tags zu löschen.
  - Sich alle Themen, wo sie einen Beitrag geleistet haben, anzeigen zu lassen.
  - Ihr Profil einzusehen, wo sie alles außer ihrem Benutzernamen auch ändern dürfen.
- Ist ein Benutzer zusätzlich Administrator, so kann dieser auch noch folgendes:
  - Beliebig alle Themen, Beiträge und Tags löschen.
  - Auf eine Benutzerverwaltung zugreifen, wo er die Daten jedes Benutzers, ausgeschlossen seines Benutzernamens, beliebig ändern kann.

Die registrierten Benutzer sowie die Themen, deren Beiträge und die Tags müssen in einer Datenbank verwaltet werden.

5.2. Implementierung

5.2.1. Entwicklungsumgebung

Zunächst einmal suchen wir uns eine Programmiersprache raus. Dafür schauen wir uns weit verbreitete Programmiersprachen im Web an. Eine der häufigsten Programmiersprachen für Web-Anwendungen ist PHP (PHP: Hypertext Preprocessor) [1]. Auch in der Zentralbibliothek wird PHP vielseitig für Web-Anwendungen benutzt. Mithilfe von XAMPP[2] benutzen wir eine MySQL[3] Datenbank zur Verwaltung unserer Daten. Damit wir auch extern die Daten der Datenbank verwalten können benutzen wir zusätzlich noch phpMyAdmin[4]. So können wir über den localhost mittels phpMyAdmin unsere Datenbank mit Tabellen füllen. Zum testen der Anwendung benutzen wir den von XAMPP nutzbaren Webserver Apache[5].

Um nicht das Rad neu zu erfinden, benutzen wir ein Framework. Hier bieten sich die bekannten Frameworks wie Laravel[6], CodeIgniter[7] oder Symfony[8] an. Schaut man sich diese genauer an, stellt man schnell fest, dass Laravel aus der Auswahl rausfällt. In Laravell gilt es allgemein als `"bad practice"` und `"poor design"` MVC oder HMVC zu benutzen. Für CodeIgniter und Symfony gibt es jeweils Erweiterungen, welche einem eine einfache Implementierung von HMVC ermöglichen. Man hat sich für CodeIgniter entschieden, da dort bereits mit MVC gearbeitet wird und sich die Einbindung von HMVC mittels einer Extension als sehr einfach erweist.

5.2.1.1. CodeIgniter

In CodeIgniter der Version 3.11 werden die URLs, welche man aufruft, standardmäßig auf das MVC-Muster aufgelöst. Dabei sieht das Schema wie folgend aus:

http://example.com/[controller-class]/[controller-method]/[arguments]

Hierbei steht

example steht für den Namen des Projekts.

controller-class steht für den Namen, welchen wir unserem Controller geben.

controller-method ist die Methode, welche wir in diesem ausführen wollen. Geben wir diese nicht an, so wird standardgemäß die `"index"` Funktion des Controllers benutzt. Daher localhost/example/main und localhost/example/main/index laufen auf die gleiche Methode im main Controller hin.

[1] Mehr zu PHP: Anhang Link 7 A
[2] Mehr zu XAMPP: Anhang Link 8 A
[3] Mehr zu MySQL: Anhang Link 9 A
[4] Mehr zu PHPMyAdmin: Anhang Link 10 A
[5] Mehr zu Apache: Anhang Link 11 A
[6] Mehr zu Laravel: Anhang Link 12 A
[7] Mehr zu Codeigniter: Anhang Link 13 A
[8] Mehr zu Symfony: Anhang Link 14 A

arguments bietet die Möglichkeit Übergabeparameter für diese Funktion zu liefern. localhost/example/main/showsomething/5/testen würde z.B. der Methode showsomething aus dem main Controller noch die Argumente `"5"` und `"testen"` übergeben.

Um nun HMVC in CodeIgniter zu benutzen, muss lediglich eine Extension installiert werden. Die URL verändert sich dann wie folgend:

http://example.com/[module-name]/[controller-class]/[controller-method]/[arguments]

Der `"module-name"` ist dann die Aufteilung in die einzelnen MVC-Triaden. Der Rest der URL-Auflösung geschieht genau wie ohne HMVC.

5.2.1.1.1. Bibliotheken

Um in View Variablen ändern zu können, benutzt man die Template Parser Class. Diese erlaubt es in den Views ersetzbare Variablen anzulegen. Diese stehen dann in geschweiften Klammern "{}". Anschließend muss man nur noch im Controller einen Array mit den zu ersetzenden Daten anlegen und diesen zum Parsen der View mitgeben. Zur Verwaltung einer Session binden wir die session Bibliothek ein. Diese erlaubt es einfach Daten in die Session einzuspeichern und zu holen, ohne dass man sich um die Verwaltung von Session und Cookies kümmern muss. Zuletzt benutzt man noch URL-helper, um einfache Links zu erstellen.

5.2.1.2. Daten-Struktur

Für unsere Web-Anwendung benötigen wir folgende Datenbank-Struktur:

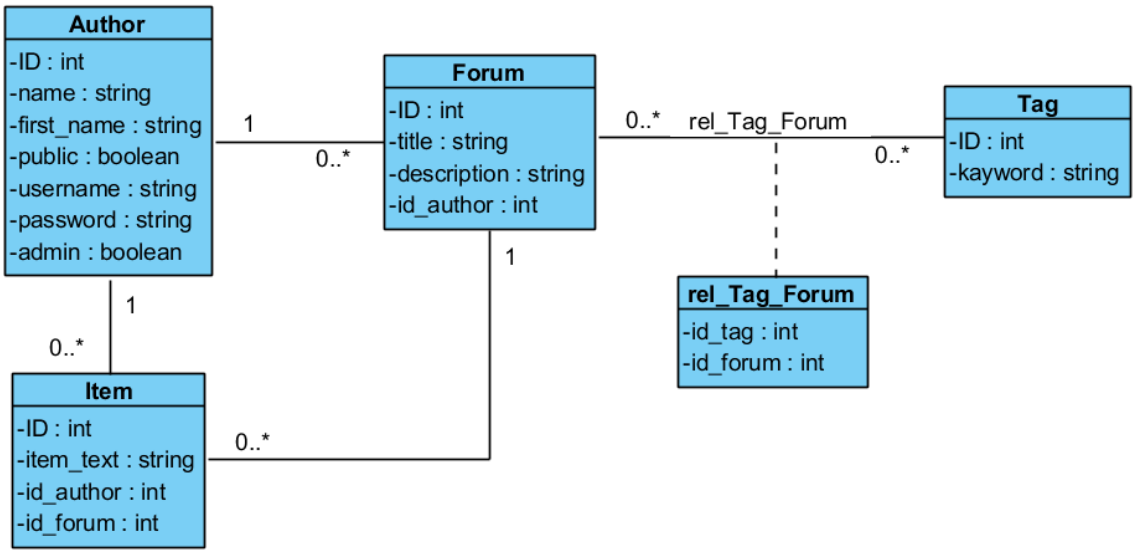

Abbildung 5.1.: Datenbank-Struktur

5.2.2. Entwicklung

Da es sich bei der Anwendung um eine Serverseitige Web-Anwendung handelt, ändert sich unsere Nutzung von MVC leicht ab. Vom Grundaufbau her wird immer noch Push-MVC verwendet. Da wir serverseitig arbeiten, geschieht nun keine direkte Kommunikation zwischen dem Model und der View mehr. Alle Änderungen an sowohl dem Modell als auch der View werden nun ausschließlich vom Controller übernommen.

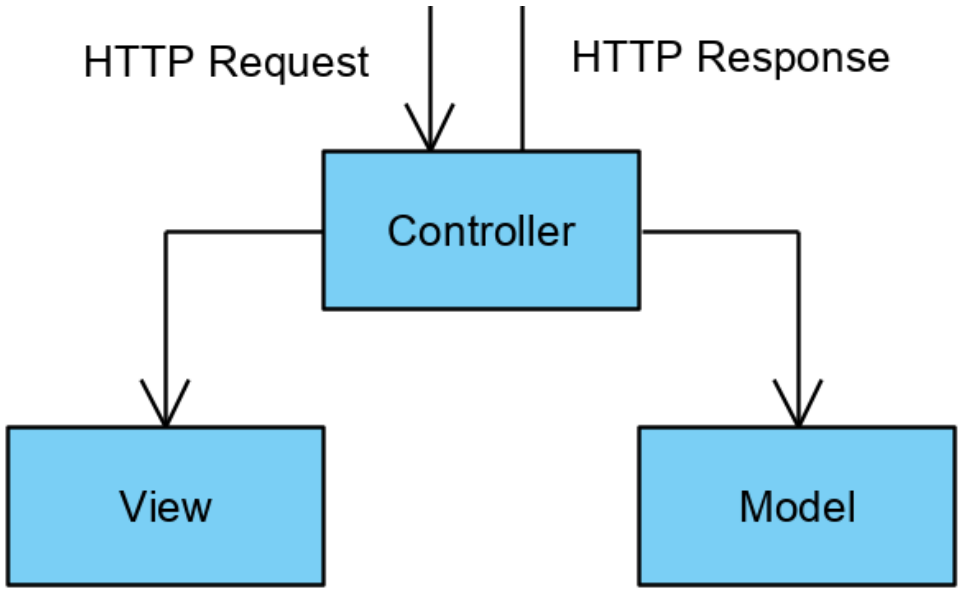

Abbildung 5.2.: Client based Web MVC

Das Programm setz sich aus 3 Modulen zusammen.

Das Site-Module: Hier befindet sich der Haupt-Controller, welcher über allen anderen steht. Ebenfalls befindet sich dort ein Navigations-Controller zur Einrichtung des Headers. Im Views-Ordner des Site-Modules sind alle Views aus denen das Grundgerüst jeder anzuzeigenden Seite geformt wird.

Das Forum-Module: Dieses Module übernimmt alle Aufgaben, welche im Zusammenhang mit den Themen und Beiträge des Forums stehen. Hier befindet sich ein Forum-Controller für alle Anfragen, das Forum-Model, mit welchem sich die besagten Daten ändern lassen, und Views zum anzeigen und erstellen von eben dieser.

Das Author-Module: Dieses Module übernimmt alle Aufgaben betreffens des Benutzers. Es besitzt ebenfalls einen Author-Controller und ein Author-Modell zur Veränderung der Benutzer-Daten. Unter den Views finden sich hier alle Möglichkeiten die Daten eines Benutzers zu verändern.

5.2.2.1. Der Site-Controller

Als erstes wurde der Site Controller umgesetzt. Dies ist der Controller, welcher bei allen Aufrufen der Seite als erstes angesteuert wird. Beim Aufruf des Site-Controllers lädt dieser zunächst alle für das weitere Vorgehen nötigen Module und bestimmt anschließend die Berechtigungslevel des momentanen Benutzers.

Listing 5.1: Site Konstruktor

```
$this->load->module('forum/forum');
$this->load->module('author/author');
$this->load->module('site/navigation');
$this->load->library('form_validation');
$this->load->helper('url');
$this->isloggedin = ($this->author->isLogin());
$this->isadmin = ($this->author->isAdmin($this->isloggedin));
```

Anschließend wird die, durch die URL aufgerufene Funktion, ausgeführt. Diese sammelt dann alle für den View nötigen Daten zusammen. Um sich viel CSS-Schreiberei zu sparen, wird hierfür eine Layout-View verwendet. Diese sieht wie eine ganz normale HTML-Datei aus, mit einem <head> und einem <body> zusammengebaut. Der body baut sich wie folgend zusammen:

Listing 5.2: Body des Layouts

```
<body>
{header}
<div id="container">
  <h1>{heading}</h1>
  <div id="body" style="background-color:<?php␣echo␣$color␣?>;">
    {content}
  </div>
</div>
{footer}
</body>
```

Um gewisse Teile des Layouts austauschen zu können, parsen wir die in `{}` geschriebenen Informationen erst später in die View hinein.

Zurück zum Controller. Dieser speichert einfache String Informationen wie Heading und Title in einen Data Array. Den Footer, welcher Buttons enthält, laden wir als View in diesen Array hinein. Da sich unser Header bedingt daran, ob wir eingeloggt und/oder Administrator sind ändert, lagern wir das Laden dieser View in einen Navigations-Controller aus. Dieser gibt bei Aufruf seiner header() Funktion den richtigen Header wieder. Danach fehlt unserem Data Array nur noch der Content. Für diesen startet der Controller jeweils eine Anfrage an einen oder mehrere zuständige Controller. Bedarf einer der Controller an Daten zum Beisiel einer `forum_id` so wird diese in der URL mitgegeben und vom Site-Controller einfach beim Aufruf des Unter-Controllers weitergeleitet. Hat die Funktion alle Daten im Array zusammen, müssen diese nur noch mithilfe des Parsers im Layout ersetzt werden.

Listing 5.3: Index Funktion im Site-Controller

```
function index() {
  $data = array(
    'header'   => $this->navigation->header($this->isloggedin,
       $this->isadmin),
    'footer' => $this->load->view('footer', '', TRUE),
    'heading' => 'Gefundene Forum-Eintraege',
    'title' => 'Forum',
    'color' => '#F5B7B1'
  );
  $data['content'] = $this->forum->getAllForum($this->isloggedin,
     $this->isadmin);
  $this->parser->parse($this->config->item('default_layout'),
     $data);
}
```

Einige Funktionen weichen jedoch von diesem Muster ab. Darunter fallen Beispielsweise alle Funktionen, welche für das Löschen von Elementen zuständig sind. Diese machen meist einen Aufruf an einen, für das zu löschende Element zuständigen, Controller und laden anschließend die bereits offene Seite neu. Ebenfalls weicht die logout() Funktion davon ab, da diese einen ohne Umweg zurück zur Startseite weiterleitet.

5.2.2.2. Der Forum-Controller

Der Forum Controller besitzt grundlegend 3 Arten von Funktionen. Diese entsprechen, ohne Update, den CRUD Operationen für Themen, Beiträge und Tags.

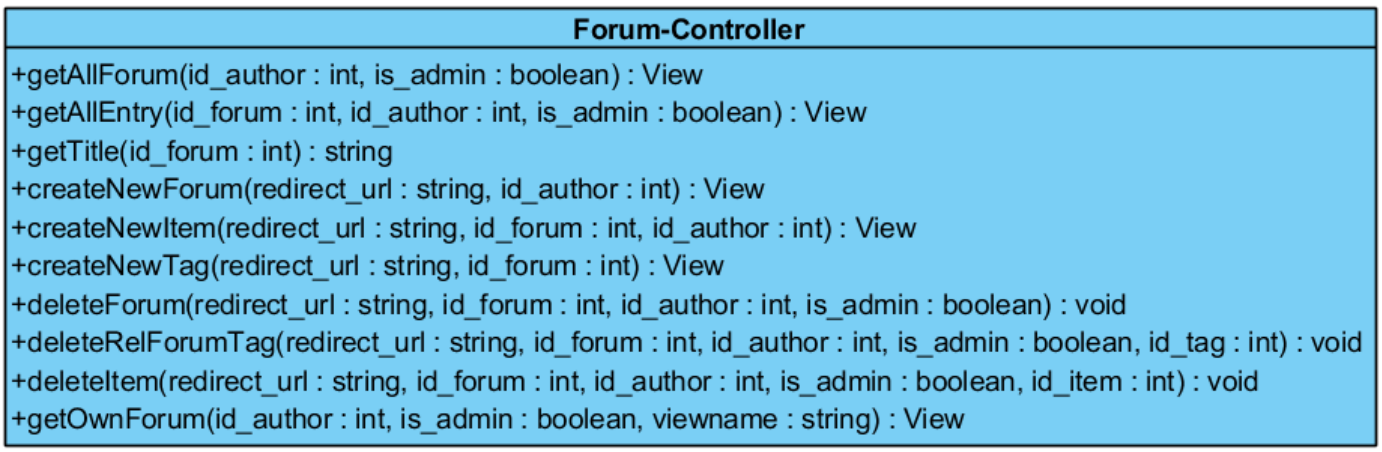

Abbildung 5.3.: Funktionen des Forum-Controllers

5.2.2.3. Das Forum-Modell

Das Forum-Modell beinhaltet, die zu den im Controller erwähnten Funktionen passenden, Datenbank-Befehle.

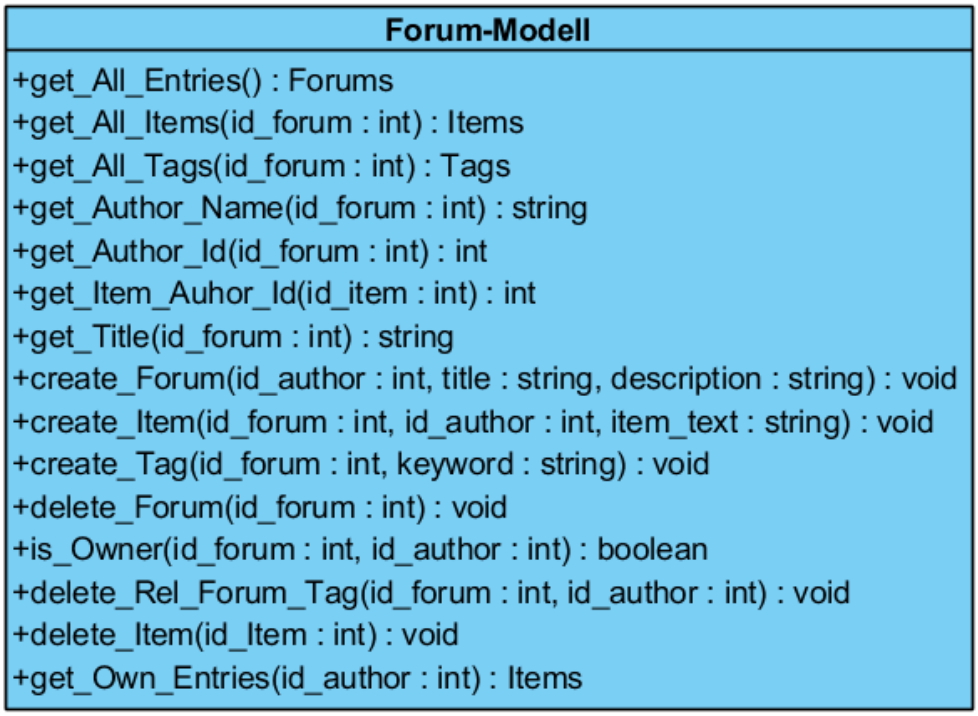

Abbildung 5.4.: Funktionen des Forum-Modells

5.2.2.4. Der Author-Controller

Der Author-Controller benutzt ebenfalls Funktionen, welche den CRUD-Operationen entsprechen. Diesmal allerdings ohne Delete, da Benutzer nicht gelöscht werden können.

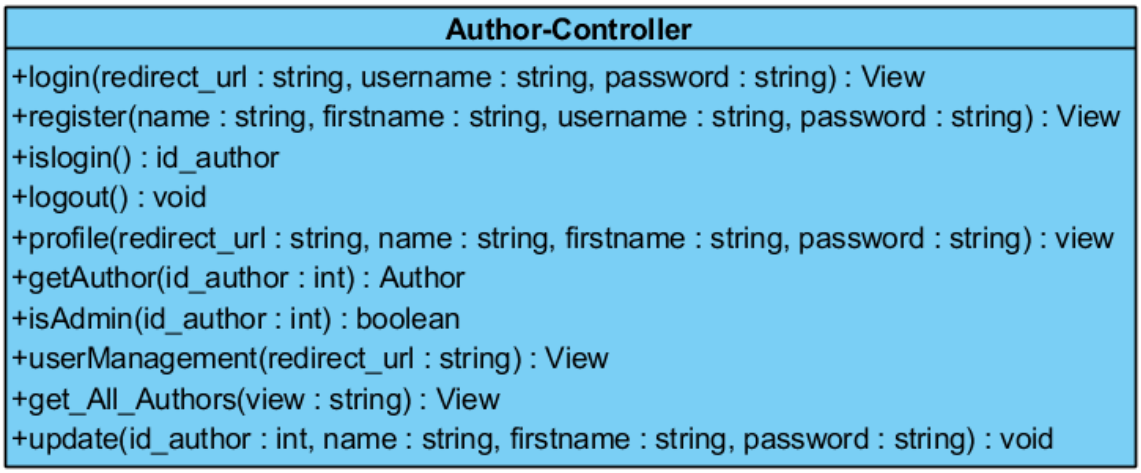

Abbildung 5.5.: Funktionen des Author-Controllers

5.2.2.5. Das Author Modell

Auch das Author-Modell beinhaltet, zu seinem Controller passende, Datenbank-befehle.

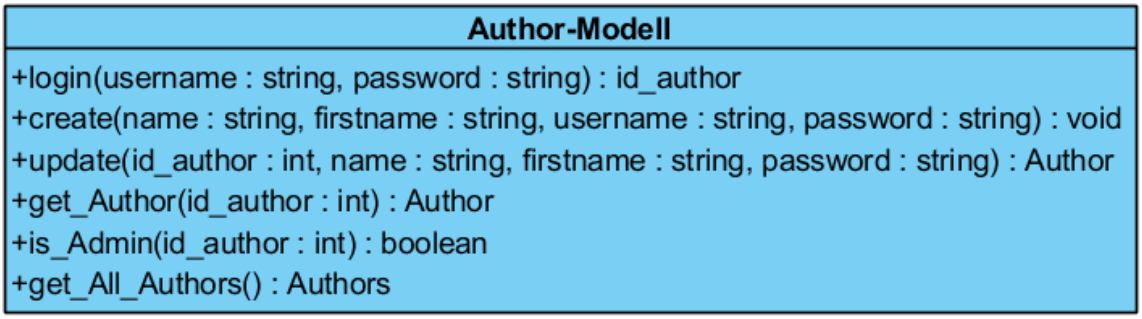

Abbildung 5.6.: Funktionen des Author-Modells

5.2.3. Endprodukt

Abbildung 5.7.: Homepage

Das Bild 5.7 zeigt die Homepage der fertigen Anwendung. Header und Footer sind erkennbar farblich abgehoben. Der Teil in der Mitte stammt vom Forum-Controller und wurde in die Layout View eingefügt.

6. HMVC versus MVC

6.1. Theorie

Wie wir bereits in Kapitel 3 gesehen haben, bietet MVC einige Vorteile in der Web-Entwicklung. Schauen wir uns nun an, was sich bei HMVC geändert hat.

	MVC	HMVC
Wiederverwendbarkeit der Komponenten	+	Bei HMVC erhöht sich die Wiederverwendbarkeit noch einmal enorm. Von nun an können nicht nur die drei Teile von MVC als Pakete wiederverwendet werden, sondern einzelne Bereiche des Programmes, entweder als ganze MVC-Triade oder deren Einzelteile können beliebig wiederverwendet und kombiniert werden. So zum Beispiel kann der ganze Registrier-Bereich von einer Web-Anwendung einfach als Triade in einer anderen Web-Anwendung benutzt werden.
Austausch von Komponenten	+	Genau wie bei MVC auch können einzelne Teile einfach durch neue ausgetauscht werden. Durch die Modularität wird es nochmals einfacher gezielte Bereiche auszutauschen.
Arbeitsteilung	+	Durch die Modularität wird es ebenfalls einfacher, die Arbeit auf verschiedene Programmierer aufzuteilen. Es kann auch gezielter darauf geachtet werden, wofür mehr Zeit und wofür weniger Zeit gebraucht wird, und gegebenenfalls können dann Arbeitskräfte umverteilt werden.
Erweiterbarkeit der Komponenten	+	Die einzelnen Komponenten sind wie bei MVC beliebig erweiterbar, mit dem Unterschied, dass es nun mehr Komponenten zur Verfügung sind.
Erhöhter Arbeitsaufwand	-	Leider macht auch HMVC den erstmaligen Arbeitsaufwand nicht geringer. Für eine gute Struktur muss eben seine Zeit investiert werden. Dafür spart man sich bei der Erweiterung und Verwaltung des Projektes Zeit.
Skalierbarkeit	-	Gerade das Problem der Skalierbarkeit wird von HMVC sehr gut gelöst. Selbst bei enormen Größen des Projektes wird durch HMVC sichergestellt, dass die einzelnen Triaden immer eine überschaubare Größe behalten. Sollte eine Triade starke Ausmaße annehmen, kann immer überlegt werden mehrere Triaden daraus zu kreieren.

6.2. Praxis

Dies wird sich nun mithilfe der Web-Anwendung genauer angeschaut. Dafür nehmen wir das Beispiel, der Benutzerverwaltung. Um den Code etwas leserlicher zu gestalten kürzen wir an einigen stellen etwas ab. So wird `$this->variable` zu `variable` und

aus `$this->module->funktion()` wird `module->funktion`. Ebenfalls wird das `$data` vor jeder `['variable']` weggelassen.

Listing 6.1: Usermanagement Funktion mit HMVC

```
1 if(isloggedin && isadmin){
2  ['authors'] = author->get_All_Authors('list');
3  if(input->post('select')){
4   if (form_validation->run() === FALSE) {
5    ['selected'] = false;
6   }
7   else{
8    ['selected'] = true;
9    ['author'] = author->getAuthor(input->post);
10  }
11 }
12 else if(input->post('update')){
13  ['selected'] = true;
14  ['author'] = author->getAuthor(input->post);
15  if (form_validation->run()) {
16   author->update(input->post);
17   ['author'] = author->getAuthor(input->post);
18  }
19 }
20 else{
21  ['selected'] = false;
22 }
23 }
24 if(['selected']){
25  ['forums'] = forum->getOwnForum(input->post, false, 'list1');
26 }
27 ['content'] = load->view('userManagement', $data, TRUE);
28 parser->parse(config->item('default_layout'), $data);
```

Vergleichen wir nun mit der Umsetzung des ganzen mittels MVC verändern sich 5 Stellen. Dies sind die Zeilen 2, 9, 14, 17 und 25.

Listing 6.2: Usermanagement Funktion mit HMVC

```
['authors'] = author->
    get_All_Authors('list');

['author'] = author->getAuthor(
    input->post);

['author'] = author->getAuthor(
    input->post);

['author'] = author->getAuthor(
    input->post);

['forums'] = forum->getOwnForum
    (input->post, false, 'list1
    ');
```

Listing 6.3: Usermanagement Funktion mit MVC

```
['authors'] = load->view('list'
    , array('authors' =>
    MVCModel->get_All_Authors()
    ), TRUE);

['author'] = load->view('author
    ', array('author' => (array
    )MVCModel->get_Author(input
    ->post)), TRUE);

['author'] = load->view('author
    ', array('author' => (array
    )MVCModel->get_Author(input
    ->post)), TRUE);

['author'] = load->view('author
    ', array('author' => (array
    )MVCModel->get_Author(input
    ->post)), TRUE);

['eintraege'] = MVCModel->
    get_Own_Entries(input->post
    );
['id_author'] = input->post;
['is_admin'] = false;
['forums'] = load->view('list1'
    , $data, TRUE);
```

Hier wird deutlich, dass bei MVC, die Arbeit, welche bei HMVC noch in den Unter-Cotrollern lag, nun vom Haupt Controller selbst übernommen werde muss. Am besten wird dies von der letzten Veränderung repräsentiert, da dort mehr Daten geholt werden müssen, um mit MVC das gleiche zu erreichen wie vorher mit dem Unter-Controller. Somit wird klar warum wir, mit der Modularisierung von HMVC, eine bessere Skalierbarkeit und Erweiterbarkeit erreichen.

7. Schlussfolgerung

7.1. Zusammenfassung

Ziel dieser Seminararbeit war es herauszufinden, in wie weit HMVC, einem Nachfolger von MVC, die Nachteile seines Vorgängers ausbessert, und in wie weit HMVC als Design-Pattern taugt. Dafür wurde sich zunächst mit MVC sowie seinen Vor- und Nachteilen beschäftigt. Anschließend daran wurde HMVC genauer unter die Lupe genommen. Nachdem klar war, wie ein Web-Projekt basierend auf HMVC auszusehen hat, galt es dies anhand eines Beispiels zu testen. Dafür wurde eine Web-Anwendung in Form eines Forums entwickelt. Diese weist nun Code Beispiele auf, wo der Unterschied zwischen MVC und HMVC deutlich hervorsticht. Anhand dieser Unterschiede lässt sich veranschaulichen, in wie weit HMVC eine bessere Lösung darstellt. HMVC bietet nicht nur eine höhere Modularität, die für mehr Möglichkeiten der Wiederverwendbarkeit und Erweiterbarkeit sorgt. Sondern Probleme von MVC wie die Skalierbarkeit werden durch die Aufteilung in kleinere und überschaubare Triaden gelöst. Zwar erhöht sich einmalig der Programmieraufwand, um die Struktur von HMVC aufzustellen, aber langfristig gesehen bringt, HMVC gegenüber MVC viele Vorteile.

7.2. Aussicht

Basierend auf den herausgefundenen Daten kann gesagt werden, dass Web-Projekte, welche auf HMVC aufbauen, deutliche Vorteile gegenüber Web-Projekten, welche nur auf MVC aufgebaut sind, haben. Also sollten alle größeren Web-Projekte in Zukunft nicht mehr auf MVC, sondern auf fortschrittlicheren Design-Patterns basieren. Zum Beispiel könnten sie auf HMVC aufgebaut werden. Wichtig ist hierbei zu beachten, dass diese Arbeit sich nur damit beschäftigt, wieso HMVC eher als MVC benutzt werden sollte. Sich alle momentan existierenden Erweiterungen und Alternativen von MVC anzuschauen und zu vergleichen wäre eine Aufgabe, welche den Rahmen dieser Seminararbeit deutlich sprengen würde.

A. Anhang

[1]Website Oder Webanwendung - https://byteq.com/de/b/webseite-webanwendung-unterschied - [Online; Entnommen: 4.1.2020]

[2]Jason Cai, Ranjit Kapila and Gaurav Pal: HMVC: The layered pattern for developing strong cliet tiers - https://www.javaworld.com/article/2076128/hmvc--the-layered-pattern-for-developing-strong-client-tiers.html - [Online; Entnommen: 4.1.2020]

[3]Hamza Ouaghad: Alternatives to HMVC with Laravel - https://coderwall.com/p/itnqyq/alternatives-to-hmvc-with-laravel - [Online; Entnommen: 4.1.2020]

[4]Facebook: MVC Does Not Scale, Use Flux Instead [Updated] - https://www.infoq.com/news/2014/05/facebook-mvc-flux/ - [Online; Entnommen: 6.1.2020]

[5]codeigniter-modular-extensions-hmvc - https://bitbucket.org/wiredesignz/codeigniter-modular-extensions-hmvc/src/codeigniter-3.x/ - [Online; Entnommen: 4.1.2020]

[6]Struts 2 - Architecture - https://www.tutorialspoint.com/struts_2/struts_architecture.htm - [Online; Entnommen: 4.1.2020]

[7]Was ist PHP? - https://www.php.net/manual/de/intro-whatis.php - [Online; Entnommen: 4.1.2020]

[8]Wikipedia: XAMPP - https://de.wikipedia.org/wiki/XAMPP - [Online; Entnommen: 4.1.2020]

[9]MySQL - https://www.mysql.com/de/ - [Online; Entnommen: 4.1.2020]

[10]Bringing MySQL to the web - https://www.phpmyadmin.net - [Online; Entnommen: 4.1.2020]

[11]Apache HTTP Server Project - https://httpd.apache.org - [Online; Entnommen: 4.1.2020]

[12]The PHP Framework for Web Artisans - https://laravel.com - [Online; Entnommen: 4.1.2020]

[13]CodeIgniter Rocks - https://codeigniter.com - [Online; Entnommen: 4.1.2020]

[14]Symfony - https://symfony.com - [Online; Entnommen: 4.1.2020]

[15]Bild HMVC Struktur - https://stackoverflow.com/questions/5454337/mvc-vs-hmvc-for-web-application-development - [Online; Entnommen: 6.1.2020]

[16]Sam de Freyssinet: Scaling Web Applications with HMVC - https://web.archive.org/web/20160214073806/http://techportal.inviqa.com/2010/02/22/scaling-web-applications-with-hmvc/ - [Online; Entnommen: 6.1.2020]

[17]Wikipedia: Hierarchical model–view–controllerhttps://en.wikipedia.org/wiki/Hierarchical_model\T1\textendashview\T1\textendashcontroller - [Online; Entnommen: 6.1.2020]

[18]Wikipedia: Model View Controller - https://de.wikipedia.org/wiki/Model_View_Controller - [Online; Entnommen: 6.1.2020]

[19]HMVC – Frameworks in PHP und deren Probleme - https://net-developers.de/2010/09/07/hmvc-frameworks-in-php-und-deren-probleme/ - [Online; Entnommen: 6.1.2020]

[20]Setup HMVC with Codeigniter 3 - https://www.roytuts.com/setup-hmvc-with-codeigniter-3/ - [Online; Entnommen: 6.1.2020]

[21]HMVC: An Introduction and Application - https://code.tutsplus.com/tutorials/hmvc-an-introduction-and-application--net-11850 - [Online; Entnommen: 6.1.2020]

[22]Bild MVC - https://de.wikipedia.org/wiki/Datei:ModelViewControllerDiagram2.svg - [Online; Entnommen: 6.1.2020]

[23]Fragen an die HMVC Entwickler - https://web.archive.org/web/20050205080537/http://www.javaworld.com/javaworld/jw-09-2000/jw-0908-letters.html - [Online; Entnommen: 4.1.2020]

[24]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (GOF): Design Patterns - Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software, 1.Auflage mitp Verlags GmbH & Co. Kg, Rheinbreitbach 2015