Winterprüfung 20xx

Fachinformatiker für Anwendungsentwicklung

Automatisierter Mechanismus zum Laden/Verifizieren von Softwarekomponenten

Abgabedatum: xx.xx.20xx

Designing Education
Connecting People

Anschrift Praktikumbetrieb

Inhaltsverzeichnis

Listingverzeichnis

Diagrammverzeichnis

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf und Aufbau des IHK–Abschlussprojektes **„Automatisierter Mechanismus zum Laden/Verifizieren von Softwarekomponenten“**.

1.1 Projektumfeld

Auftraggeber ist eine Werbeagentur mit Schwerpunkt Onlinepräsenz. Zum Kerngeschäft gehört das Erstellen, Optimieren und Warten von Webseiten, das Entwickeln von Forensoftware sowie das Programmieren von Logistik- und Verwaltungslösungen. Zurzeit arbeitet die Agentur an der Diversifikation der Angebotspalette. Hierzu gehört auch ein Framework zum Einbinden von ERP-Systemen, welches 20xx released wird. Die Agentur beschäftigt noch einen weiteren Praktikanten, dessen Projekt sich klar von meinem abgrenzt.

1.2 Projektziel

Das Ziel des Projekts ist die Möglichkeit funktionale Prozesse sowie die dazu gehörenden Daten und Oberflächen, die Softwaremodule[1] (Module) genannt werden, automatisiert in das Trägersystem zu laden (ModuleLoader[2]). Aus der Sicht des Entwicklers sollen die Module mit wenigen Schritten in das System eingetragen werden können.

Beim Laden ins System muss ebenfalls sichergestellt werden, dass die bereitstellende Instanz die Lizenz zum Verwenden des Softwaremoduls besitzt. Sollte diese Lizenz fehlen, wird das Modul nicht beim Aufrufer angezeigt.

Der Vorteil der Modularisierung aller nicht zur Trägersoftware gehörenden Softwarekomponenten ist die Möglichkeit diese unabhängig vom Gesamtsystem zu planen, zu programmieren und zu testen. Dies soll die Qualität des entstandenen Programmcodes steigern und die Wiederverwendbarkeit gewährleisten.

1.3 Projektbegründung

Im Rahmen der Erstentwicklung einer Platform-as- a-Service (PaaS[3]) Software (folgend Trägersoftware) sollen zur generellen Wartbarkeit, Pflege und Fehlerlosigkeit des Bestandcodes Softwaremodule eingesetzt werden. Dem Entwickler soll es leicht gemacht werden, für sich vollständige, fehlerfreie und nach außen wohl definierte Softwaremodule zu schreiben. Um dieses Ziel zu erreichen wurde entschieden, ein System zum automatisierten Laden und Validieren / Verifizieren von Softwaremodulen zu entwickeln. Die benötigten Schritte, um das Modul bei einem Projekt einzubinden, sollen dabei einfach und unkompliziert ablaufen, so dass der Entwickler des Moduls sich nicht exakt mit den Prozessen des zugrunde liegenden Systems auskennen muss. Die Kombination von Modulen und ModuleLoader wird die Entwicklungszeit sowie generell die Qualität der Arbeit verbessern, da sich der Programmierer auf die Entwicklung des Moduls konzentrieren kann.

1.4 Projektschnittstellen

Die Hauptschnittstellen des Projekts sind die zur Verfügung gestellten lizenzierten Module im Gesamtsystem. Neben einer direkten Schnittstelle für Layoutentwickler stellt das Projekt Schnittstellen zum SiteManager[4], ModuleController[5], PageController [6], dem rollenbasierten Rechtesystem sowie zur Datenbank zur Verfügung. Auch Anwender des Backends[7] erhalten eine Schnittstelle zum Projekt, indem das Backend Informationen zu den lizenzierten / ablaufenden und abgelaufenen Modulen liefert. Neben diesen Informationen stellt das Backend der Module auch die Konfiguration ebendieser für alle Anwender zur Verfügung. Es wird ebenfalls eine Schnittstelle zu anderen Programmteilen bieten, die zur Projektzeit noch nicht entwickelt wurden.

Das Projekt wurde von Herr Duncan genehmigt und in der Abschlussphase auf seine korrekte Funktion hin überprüft. Jegliche Ausrüstung für das Projekt, ob Software oder Hardware, wurden vom Praktikumsbetrieb gestellt.

1.5 Projektabgrenzung

Für die korrekte Funktionsweise des in der Projektarbeit beschriebenen Systems werden weitere Programmteile benötigt. Von folgenden Programmteilen grenzt sich mein Projekt ab: SiteManager, PageController, ThemeLoader[8], Themes [9], Rechtesystem, Domains sowie weiteren Kernelementen der Trägersoftware. Darunter fallen unter anderem abstrakte Klassen[10] oder Interfaces[11] / Traits[12]. Auch wenn sich mein Projekt von den beschriebenen Komponenten abgrenzt, ist es möglich, dass diverse Elemente angepasst werden. Dies ist notwendig, denn der ModuleLoader ist ebenfalls ein Kernsystem der Trägersoftware, weshalb er für sich isoliert gesehen nicht funktionieren würde. Das Rechtesystem wird von einem weiteren Praktikanten entwickelt, der zur gleichen Zeit sein Praktikum absolviert wie ich.

1.6 Allgemeine Bemerkungen

Im Folgenden finden sich diverse Quellcode-Listings. Der Code wird nicht in Gänze in der Projektdokumentation enthalten sein, weil dieser der betrieblichen Geheimhaltung unterliegt.

Die Dokumentation verwendet zur Benennung der Klassen und Methoden die gleichen englischen Begriffe in CamelCase-Schreibweise wie im Projekt auch.

2 Projektplanung

2.1 Projektphasen

Die Anforderungen an die Projektdokumentation erlauben nicht wesentlich weniger oder mehr als 70 Stunden für die Durchführung des Projekts. Die 70 Stunden werden in aufeinander folgende Phasen unterteilt, um eine genauere zeitliche Zuordnung zu finden. Gering-fügige zeitliche Abweichungen der einzelnen Phasen werden im ***Kapitel 8.1 Soll-/Ist Vergleich S.14*** erläutert. Eine Übersicht der Phasen befindet sich im ***Anhang I.I Detaillierte Projektphasen mit Zeitplanung S. I.***

2.2 Ressourcenplanung

Das gesamte Projekt wird am Arbeitsplatz durchgeführt. Dort ist alle benötigte Software und Hardware vorhanden.

Hardware:

- Lenovo S540 Thinkpad
- Flachbildschirm
- Tastatur
- Maus

Software:

- Windows 10 Pro
- Atom-Editor als Entwicklungsumgebung
  - GitHub - Integration
- XAMPP Programmpaket
  - Apache Webserver 2.4.39
  - MariaDB DBMS 10.1.40
- Web-Browser
- Draw Diagrams 1.1.14
- Microsoft Office 365

Verwendete Programmiersprachen:

- PHP 7.3.5, HTML 5, CSS 3, SQL (DDL, DML)

Das Projekt wurde von Herrn Duncan als Lead-Developer unterstützt.

2.3 Entwicklungsprozesse

Die Werbeagentur arbeitet nach dem Prinzip des evolutionären Prototypings[13]. Das Ziel dieses Prinzips ist, die Prototypen schon mit sehr weit ausgearbeiteten Anforderungen zu entwickeln. Die Grundfunktionen sollen dabei vollständig abgebildet sein. Zukünftige Anforderungen von Kunden können dann schnell und hochwertig nachgeliefert werden.

Das Ergebnis ist ein qualitativ hochwertiges Produkt, das besser auf die Ansprüche des Kunden zugeschnitten ist.

3 Analysephase

3.1 Ist-Analyse

Wie bei den bisher beschriebenen Punkten Projektziel, Projektbegründung und Projektschnittstellen ist es nur mit viel Aufwand möglich, Erweiterungen für die Software zu entwickeln. Der aktuelle Stand der Trägersoftware besitzt weder Module noch die Möglichkeit diese kontrolliert ins System zu implementieren. Die Validierung findet ebenfalls nicht statt. Die einzelnen Softwarepakete müssen nach aktuellem Stand für die kundenspezifische Software stets komplett neu entwickelt werden, wodurch die Entwicklungszeit und Wiederverwendbarkeit leidet.

3.2 Wirtschaftlichkeit

3.2.1 Make or Buy-Entscheidung

Der ModuleLoader sowie das Framework für den er entwickelt wird sind so speziell, dass es hierfür keine andere Software auf den Markt gibt, die den Anforderungen von evolved-coding entsprechen würden. Deswegen ist die Überlegung etwas Passendes einzukaufen nicht gegeben, aufgrund dessen nur die Eigenentwicklung Sinn macht.

3.2.2 Projektkosten

Bei den Projektkosten werden die Personalkosten herangezogen. Es wird von einem tariflichen Bruttogehalt von 1014,00€ nach einem Auszubildenden Fachinformatiker für Anwendungsentwicklung im dritten Lehrjahr ausgegangen. Bei einer Regelarbeitszeit von 8 Stunden täglich, an 220 Tagen im Jahr, ergibt sich ein Stundenlohn von 6,91€. Mit folgender Berechnung kann man dieses Ergebnis nachvollziehen:

$$8\ \text{h/Tag} \times 220\ \text{Tage/Jahr} = 1760\ \text{h/Jahr}$$

$$1014{,}00\ €/\text{Monat} \times 12\ \text{Monate/Jahr} = 12168{,}00\ €/\text{Jahr}$$

$$12168\ €/\text{Jahr} \div 1760\ \text{h/Jahr} \approx 6{,}91\ €/\text{h}$$

Von der IHK vorgegeben darf das Projekt insgesamt 80 Stunden dauern. Herr Sowieso Stundenlohn beziffert sich auf 102,00 € die Stunde. Folgende Tabelle stellt die Kosten übersichtlich dar:

Vorgang	Stunden	Mitarbeiter	Kosten Mitarbeiter/h	Gesamtkosten
Projektzeit	80	Praktikant	6,91 €	552,80 €
Fachgespräch	6	Lead Developer	102,00 €	612,00 €
Code Review & Abnahme	3	Lead Developer	102,00 €	306,00 €
Gesamtsumme				**1470,80 €**

Kostenübersicht

3.2.3 Amortisationsdauer

Durch die Entwicklung und Implementierung des ModuleLoaders entstehen Kostenvorteile, die zukünftige Arbeiten mit dem Trägerframework erleichtern. Das Projekt wird komplett intern verwendet und stellt eine weitere Grundlage für das Framework zur Verfügung. Da weder Umsatz noch Verwendungsdauer bekannt sind kann keine ordentliche Amortisationsberechnung durchgeführt werden. Eine Kosten-Nutzwert-Analyse kann nicht erstellt werden. Zukünftige Projekte wie einzubindende Module profitieren durch die Realisierung des Projekts, da das Projekt die technischen Schnittstellen für weitere Entwicklungen zur Verfügung stellt.

Mit Abschluss der Entwicklungen werden die Projektkosten für das Framework (in dem der ModuleLoader einfließt) anteilig berechnet werden.

3.3 Lastenheft/Fachkonzept

Ein klassisches Lastenheft eines Auftraggebers existiert bei diesem Projekt nicht. Die Kernkonzepte des Projekts wurden in einem Fachgespräch ermittelt und festgehalten.

Es ist ein System für das Trägerframework zu entwickeln, was in der Lage sein soll:

- Module und deren Methoden unabhängig vom Gesamtmodul als Teilausschnitte auf jeder Seite des Projekts anzeigen zu lassen
- Nur lizenzierte Module sollen angezeigt werden – die Prüfung der Lizenzierung übernimmt dabei der ModuleLoader während er vom SiteManager instanziiert wird
- Die Instanziierung des ModuleLoaders und somit die Prüfung soll bei jedem Seitenaufruf erfolgen
- Die Information, welche Module lizenziert sind, sollen anderen Programmteilen zur Verfügung stehen
- Module dürfen nicht ohne Warnung beim Nutzer ausgeblendet werden, deshalb soll es die Möglichkeit geben, einen Warnzeitraum zu definieren
- Module, welche im Warnzeitraum liegen, sollen als Schnittstelle anderen Programmteilen zur Verfügung gestellt werden
- Sollten Module eigene Datenbankinformationen besitzen, dürfen diese trotz abgelaufener Lizenzierung nicht verloren gehen
- Die Information, welche Module wegen abgelaufener Lizenzierung ausgeblendet werden, sollen ebenfalls anderen Programmteilen zur Weiterverwendung zugänglich gemacht werden
- Es soll möglich sein, die Verwaltungsinformationen von Modulen in der evolved-coding Datenbank zu erstellen, zu lesen, zu aktualisieren und zu löschen
- Die Möglichkeit, neue Modulabschnitte in einer vorhandenen Seite zu integrieren, soll schnell und einfach für die Entwickler realisierbar sein

Die Projektlogik soll mit PHP geschrieben werden. Das Benutzerinterface soll nur soweit erstellt werden, dass die Funktionalität des ModuleLoaders getestet werden kann sowie die Modulinformationen wie beschrieben verwaltet werden können.

4 Entwurfsphase

4.1 Zielplattform

Der ModuleLoader soll als Teil einer Webanwendung[14] geschrieben werden. Damit bleibt das Gesamtsystem unabhängig vom Betriebssystem. Jedes Gerät mit Browserunterstützung kann das System verwenden, egal ob Windows, Linux / Unix, MacOS, iOS oder Android. Das System kann von vielen Plattformen international verwendet werden – sofern eine Internetverbindung existiert. Die Hauptberechnungen finden auf dem Webserver[15] statt und nicht auf dem Client des Nutzers. Deswegen können im Firmenumfeld auch einfache Thin-Clients[16] verwendet werden, die relativ preisgünstig und langlebig sind. Wie bei der Nennung der Betriebssysteme schon erkennbar ist, werden ebenfalls moderne mobile Geräte unterstützt.

4.2 Architekturdesign

Im Wesentlichen soll das Architekturdesign des Projekts die nicht funktionalen Anforderungen der Trägersoftware erfüllen – hohe Wartbarkeit, Erweiterbarkeit, Wiederverwendbarkeit, Einfachheit und Testbarkeit.

Um diese Anforderungen zu erfüllen wurde vereinbart, dass bis auf wenige Kernelemente jedes Softwareprojekt modular aufgebaut wird. Diese Module sind wiederum selbst auf kleinste unabhängige Einheiten zurück zu führen. Durch Namenskonventionen für das Projekt wird sichergestellt, dass es zu einer besseren Wartbarkeit bei den Schnittstellen kommt. Der ModuleLoader, der in dieser Projektdokumentation beschrieben wird, stellt dabei nur ein weiteres Kernelement unter vielen dar.

Die Architektur des ModuleLoaders kann nur unter Einbeziehung weiterer Elemente beschrieben werden. Abgrenzende Komponenten sind in ***1.5 Projektabgrenzung S.2*** beschrieben.

Im Wesentlichen instanziiert der SiteManager den ModuleLoader als Singleton Objekt. Während der Instanziierung prüft dieser, welche Module für welche Domain[17] gültig sind. Nicht nur die Gültigkeit der Lizenzierung muss geprüft werden, sondern auch, ob diese Module schon für das verwendete Theme umgesetzt wurden.

Es entsteht ein Array mit Modulen-Referenzen, die lizenziert und für das Theme verfügbar sind. Nach der Instanziierung des ModuleLoaders prüft der SiteManager die angeforderte Seite sowie deren Methoden und lädt den passenden PageController. Sollen gewisse Module an einer Stelle geladen werden, wird dort ein Verweis benötigt, welches Modul mit welcher Methode an dieser Stelle stehen soll. Der ModuleLoader wird erneut aufgerufen, prüft ob entsprechendes Modul auch in dem vorher verifizierten Array liegt und gibt über den ModuleController des spezifischen Moduls die URL (Uniform Resource Locator) zurück. Das Ergebnis ist ein Templatepartial des Moduls, das man neben weiteren Partials auf jeder Seite einbinden kann. Der PageController gibt dem SiteManager das vollständige Template zurück. Technisch wird dies über dynamische Einbindungen sowie über Reflexionen[18] gelöst.
Das Konzept, wie die verschiedenen Elemente zusammenarbeiten, wird im ***Anhang III.V Komponentendiagramm S. X*** übersichtlich veranschaulicht.

Um den zeitlichen Ablauf der Komponenten und Entitäten abzubilden wurde ein zusätzliches Sequenzdiagramm erstellt, welches in ***Anhang III.VI Sequenzdiagramm S. XI*** die zeitliche Abfolge besser darstellt als das Komponentendiagramm.

4.3 Datenmodell

Um die verwendeten Entitäten des Projekts zu veranschaulichen wurde ein ER-Diagramm entwickelt, welches in ***Anhang III.I ER-Diagramm S. VI*** dargestellt wird.

Dieses ER-Diagramm wurde dann in ein Domänendiagramm überführt, dargestellt in ***Anhang III.II Domänendiagramm S. VII***.

Um die ermittelten Entitäten des ER-Diagramms in die Datenbank zu übertragen, wurde ein physikalisches Datenmodell entwickelt, welches in ***Anlage III.III Physikalisches Datenmodell S. VIII*** dargestellt wird. Um die in dem Projekt verwendeten Klassen darzustellen, wird ein UML Klassendiagramm verwendet.

Dabei kann allerdings auf Klassen, die sich vom Projekt abgrenzen, aufgrund des Betriebsgeheimnisses nicht weiter eingegangen werden. Es stellt die verwendeten Klassen des Projekts dar, ohne aber auf Variablen oder Methoden einzugehen. Das UML Klassendiagramm ist in ***Anlage III.IV Klassendiagramm S. IX*** dargestellt.

4.4 Test-Konzept

Tests wurden manuell ausgeführt. Mit dem Fertigstellen einzelner Funktionen wurde deren interne Logik getestet. Unter anderem wurden typische Eingabeparameter nach dem Prinzip der Äquivalenzklassenbildung getestet. Dies bedeutet, dass eine endliche Anzahl Klassen „ähnlicher Werte“ zur Prüfung herangezogen werden. Geprüft werden aus dieser Menge einzelne Exemplare jeder Klasse. Neben diesen Tests wurde auch eine Grenzwertanalyse durchgeführt. Hierbei werden nur die Werte der Grenzen der Äquivalenzklassen getestet. Danach wurden die Funktionen auf einen größeren Programmkontext getestet und ob sie mit anderen Funktionen das erwartbare Ergebnis liefern. Sollten grundlegende Veränderungen gemacht worden sein, die sich auf andere Programmteile auswirken können, wurden diese ebenfalls auf ihr erwartetes Verhalten getestet.

Um die letztliche Funktion des ModuleLoaders zu testen, wurden einfache Template-Ausschnitte (Templatepartial [19]) erstellt, die dann testweise auf einer Seite eingebunden wurden. Ein Templatepartial kann alle Funktionen darstellen, die für eine moderne Webanwendung benötigt werden, zum Beispiel ein Formular zum Erzeugen von Dateneingaben oder das Lesen von Informationen. Es wurden Beispielmodule erstellt, die einfache HTML-Texte darstellen. Durch das erfolgreiche Einbinden dieser Templates kann die Funktionsweise des automatisierten Ladens eindeutig verifiziert werden. Ebenfalls wurde getestet, dass Module deren Lizenz abgelaufen sind, nicht mehr geladen werden. Ein Nutzer hat auch die Möglichkeit, benutzerdefinierte Module zu erstellen. Diese liegen allerdings an einem anderen Speicherort. Auch das korrekte Einbinden dieser Module wurde getestet.

4.5 Pflichtenheft

Bereits in ***3.3 Lastenheft/Fachkonzept S. 5*** wurde erläutert, dass kein klassisches Lastenheft für das Projekt existiert. Ebendies trifft auch auf das Pflichtenheft zu. Die geforderten Anforderungen des Lastenhefts wurden mit den Erläuterungen und Entwürfen in 5.1 bis 5.4 sowie den dazugehörigen Diagrammen beschrieben. Herr Duncan als Lead Developer des Projekts hat diese Ergebnisse als gut empfunden und die geschilderten Ausführungen bestätigt.

5 Implementierungsphase

5.1 Erstellen der Projektstruktur

In der ersten Implementierungsphase werden grundlegende Projektstrukturen angelegt. Eine der Projektanforderungen ist unter anderem, neue Verzeichnisse, Dateinamen sowie Methoden vereinheitlicht zu erstellen.

Um den ModuleLoader implementieren zu können, wird ein System zur Verwaltung von Modul-Informationen benötigt. Dafür wird eine Modul-Entität sowie deren Methoden benötigt, um diese dann in der Datenbank speichern, ändern, lesen und löschen zu können.

Um die Funktion des ModuleLoaders wie in ***4.5 Test- Konzept S. 7*** beschrieben zu testen, werden diverse Beispielmodulverzeichnisse benötigt.

Die Templatepartials für jede Methode werden ebenso angelegt. Strukturiert werden diese Partials dann in einem übergeordneten Template, in welchem diese dann eingebunden werden.

Folgend wird die Verzeichnisstruktur des ModuleLoader-Verzeichnisses detailliert dargestellt.

- Controllers
  - ModuleController: Stellt Methoden zur Verfügung, über die die Entitäten verfügen, z.B. Create, Read, Update und Delete
- Entities
  - Module Entity: Hält alle Datenfelder sowie Methoden, um diese Felder zu lesen und zu schreiben
- Repositories
  - ModuleRepository: Stellt Methoden zur Kommunikation mit der Datenbank zur Verfügung
- ModuleLoader: Stellt die Logik für das automatisierte Laden der Module zur Verfügung

Die Verzeichnisstruktur eines Beispielmoduls besteht grundlegend aus der gleichen Verzeichnisstruktur wie die des ModuleLoaders. Zusätzlich kann allerdings ein Ordner mit Templates angelegt werden, sofern es sich um ein benutzerdefiniertes Modul handelt. Templates von Modulen, die nicht benutzerdefiniert sind, werden in den entsprechenden Themes gespeichert.

5.2 Implementierung der Datenstruktur

In diesem Schritt werden die benötigten Relationen für die MySQL Datenbank angelegt. Dafür wird eine Datenbank-Verbindung hergestellt, die im Falle des Scheiterns den Verbindungsversuch beendet. Anschließend wird geprüft, ob die Tabelle schon existiert, falls dem so ist wird sie gelöscht. Im nächsten Schritt wird der SQL Befehl mit dem CREATE TABLE … Befehl erzeugt. Der Vorgang wird als Listing in ***Anhang II.I Erstellen der Modul - Relation S. III*** dargestellt.

Ebenfalls muss noch die Hilfstabelle Domain_Modules erzeugt werden. Diese hält Information fest, welche Domain, welche Module lizenziert hat sowie wann diese Lizenzierung abläuft und ob das Modul benutzerdefiniert im Domainbereich liegt. Auch für die Erstellung dieser Relation wurde ein Listing zur Verfügung gestellt im ***Anhang II.II Erstellung der Module_Domains Relation Listings S. IV.***

Die Modules_Theme Tabelle ist ähnlich wie Domain_Modules Tabelle, allerdings nur mit den Fremdschlüsseln der entsprechenden Tabellen bestückt.

Die beiden Hilfstabellen wurden auch noch mit Beispieldaten befüllt, damit der ModuleLoader das System nutzen kann. Zum Entwicklungszeitpunkt existierten die Systeme, welche die Hilfstabellen mit Daten befüllen, noch nicht.

5.3 Implementierung der Funktionslogik

Folgend wird auf einige Komponenten des Projekts auszugsweise eingegangen.

5.3.1 Module Entity

Um die Module zu nutzen, werden diverse Verwaltungsinformationen benötigt. Dafür wird eine Module Entity mit folgenden Feldern erzeugt, beginnend mit dem Datentyp:

String: id, name, description, version, controller_path
Date: created_at, updated_at, expire_date
Boolean: custom, active

Diese Felder werden nicht alle benötigt um die Existenz des Moduls in der Datenbank festzuhalten. Vor allem expire_date und custom werden aber benötigt, sobald der ModuleLoader die passenden Module wieder aus der Datenbank extrahiert. Beim Erzeugen des Moduls aus der Datenbank heraus, wird überprüft, ob das Modul abgelaufen ist oder nicht. Falls es lizenziert ist, wird der ModuleController Pfad erzeugt.

Um den Controller Pfad zu bestimmen wird der Name des Modules benötigt. Im Fall eines benutzerdefinierten Moduls ist zusätzlich die aufrufende Domain erforderlich, welche vom SiteManager geliefert wird.

```
private function setControllerPath() {
    if(!$this->getCustom()) {
        $this->controllerPath = 'core\\modules\\'.$this->getName().
                                '\\controllers\\ModuleController';
    } else {
        $domain = SiteManager::getDomain();
        $this->controllerPath = 'clients\\'.$domain->getDir().'\\modules\\'
                                .$this->getName().
                                '\\controllers\\ModuleController';
    }
    return $this;
}
```

Listing 1: Controller Pfad ermitteln

5.3.2 ModuleController

Ebenfalls wichtig für die Datenhaltung im MVC-Pattern (Model-View-Controller Entwurfsmuster[20]) ist der Controller. Der ModuleController bietet unter anderem die verschiedenen Methoden des CRUDs[21] an. Eine Methode des ModuleControllers ist im **Anhang II.III Delete Methode des Module Controllers (Auszug) S. III** dargestellt.

5.3.3 ModuleRepository

Die ModuleRepository ist im Wesentlichen die Klasse, welche die Datenbankabfrage-Methoden enthält. Hier werden die SQL-Abfragen zu den CRUD-Methoden sowie alle weiteren Abfragemethoden, die mit Modulen zu tun haben, gesammelt.

Ein Beispiel wäre die delete() Methode der ModuleRepository im **Anhang II.IV**
Delete Methode der ModuleRepository S. V.

Der View Teil des MVC stellt dann das entsprechende Template zur Verfügung. Aus Platzgründen wird kein Listing gezeigt, da es einem einfachen HTML-Formular entspricht.

5.3.4 ModuleLoader

Mit den genannten Komponenten aus 5.3.1, 5.3.2 und 5.3.3. kann der projektnamensgebende Mechanismus implementiert werden. Der ModuleLoader wird nach dem Singleton Entwurfsmuster implementiert und im SiteManager (der Hauptcontroller des Trägerframeworks) instanziiert.

Das Singleton Entwurfsmuster beschreibt, dass nur eine Instanz einer Klasse existiert und dieses Objekt in der Regel global zugänglich ist. Die Singleton Implementierung wird folgend dargestellt:

```
private static $instance = null;

public static function create($department, $offset): ModuleLoader {
    if(self::$instance === null) {
        self::$instance = new static();
        ....
return self::$instance;
```

Listing 2: Instanziierung der ModuleLoader Instanz

Damit nicht weitere Instanzen des Objekts erstellt werden können, müssen anschließend noch folgende PHP-Methoden mit leerer Implementierung auf private gesetzt werden:

- `private __construct(){}`
- `private __clone(){}`
- `private __wakeup(){}`
- `private __sleep(){}`

Im nachfolgenden Schritt werden die passenden Module mit der *readMatchingModules()* Methode der ModulesRepository geladen. Diese Methode sorgt dafür, dass die passenden Module je nach Domain und Theme gewählt werden. Dies wird durch folgende SQL Abfrage erreicht:

```
$sql = 'SELECT hm.id, hm.name, hm.description, hm.created_at,
hm.updated_at, hm.version, hm.controller_path, hmd.expire_date,
hmd.custom
FROM heim_modules hm
LEFT JOIN heim_modules_domains hmd ON hm.id = hmd.module_id
LEFT JOIN heim_modules_themes hmt ON hm.id = hmt.module_id
WHERE domain_id = :domain_id AND theme_id = :theme_id';
```

Listing 3: SQL-Abfrage, Teil der readMatchingModules() Methode

Kunden bekommen nach Auftragserfassung eine URL zugewiesen, deswegen verwendet das System intern den Domainnamen gleichbedeutend mit einem Kunden, der Dienste, Anwendungen und Module lizenziert.

Die Module werden ebenfalls nach Themes gefiltert. Denn mit steigender Anzahl Themes werden nicht immer alle Modul-Layouts für jedes Theme umgesetzt. Für das Standard Theme wird dies garantiert. Wünscht sich der Kunde ein benutzerdefiniertes Theme wird das Layout vom Modul nach freien Entwicklerkapazitäten oder nach Kundenbedarf für das Theme entwickelt.

Die Rückgabe der *readMatchingModules()* Methode sind alle Module, die der Kunde lizenziert hat und für das aktuelle Theme verfügbar sind. Der ModuleLoader filtert diese nun noch mal aus:

- Nach Modulen mit gültiger Lizenz
- Module deren Lizenzierung droht auszulaufen
- Module mit abgelaufener Lizenz

5.3.5 PageController

Der PageController liefert alle Ressourcen, die der SiteManager zum Aufbau eines Dokuments benötigt. Damit ist er auch dafür zuständig die Modul-Templatepartials zur Verfügung zu stellen. Dafür ruft er die Methode getModulePartial() der abstrakten Page Klasse auf, die ihrerseits die loadModulePage() Methode des ModuleLoaders aufruft. Die Methode benötigt als Übergabeparameter den Namen des Moduls sowie die Methode. Das Ergebnis dieser Anfrage ist die URL zum Templatepartial.

```
$testModule1Partial = $this->getModulePartial('testModule1', 'read');
$this->addPartial('testModule1Partial',
                  $testModule1Partial);
```

Listing 4: Aufruf der getModulePartial() Methode

6 Qualitätsmanagement

Um die Qualität während der Entwicklung und danach zu gewährleisten wurden nach wichtigen Entwicklungsschritten manuelle Tests durchgeführt. Abschließend hat der ModuleLoader folgende Tests bestanden:

- Lade alle lizenzierten Module
- Lade alle benutzerdefinierten Module
- Lade keine abgelaufenen Module
- Alle Module, die abzulaufen drohen, müssen in einem gesonderten Array zur Verfügung stehen
- Der PageController kann Templates von lizenzierten Modulen einbinden

Um die Verwaltungsinformation für die Module zu testen wurden ebenfalls die Create, Read, Update und Delete Methoden erfolgreich getestet.

7 Dokumentation

7.1 Entwicklerdokumentation

Einen Auszug aus Entwicklerdokumentation ist zu finden unter ***Anhang IV Entwicklerdokumentation (Auszug) S. XII.***

8 Fazit

8.1 Soll-/Ist-Vergleich

Um die zeitlichen Abläufe der Komponenten besser zu vermitteln, wurde nach weiteren Möglichkeiten der Darstellung recherchiert und das UML-Sequenzdiagramm ermittelt. Die Recherche sowie Anfertigung des Diagramms hat die benötigte Zeit der Projektplanungs-phase und Dokumentationsphase leicht erhöht.

Zwei der veranschlagten Tests haben weniger Zeit in Anspruch genommen wie erwartet:

- Projektplanung: +1h
- Erstellen des Sequenzdiagramms: +2h
- CRUD Methoden testen: -2h
- Test der Modulverfügbarkeit: -1h

Die Anforderungen des Lastenhefts, **3.3 Lastenheft / Fachkonzeption S. 5,** wurden in Gänze erfüllt. Somit wird das Projektergebnis für das Unternehmen als Erfolg gewertet. Es ist sehr wahrscheinlich, dass das Projekt in Zukunft Verwendung bei der Weiterentwicklung des Trägerframeworks findet.

8.2 Aussichten

Überlegungen nach dem Abschluss des Projekts haben ergeben, dass letztlich die Struktur eines Dokuments innerhalb der Datenbank liegen wird. Ebenfalls soll der Zeitraum, in dem ein Kunde vor dem Ablaufen gewarnt wird, innerhalb einer Konfigurationsdatei als Konstante zur Verfügung stehen. Sollte die Lizenz von Modulen abgelaufen sein, muss noch deren Verbleib geklärt werden. Aus der Usability[22] (Benutzerfreundlichkeit) heraus wäre es sinnvoll, dem Nutzer die Informationen, welche Module er in der Vergangenheit verwendet hat, auch langfristig anzeigen zu lassen und falls gewünscht, diese Information endgültig zu löschen. Auch Daten, die ein abgelaufenes Modul erzeugt hat, müssen separat lösch,- oder exportierbar sein für den Kunden.

9 Quellenverzeichnis

https://www.vishia.org/SwEng/html/Dependencies.html
1.1 Divide et impera, Software modular beherrschen

https://www.ausbildung.de/berufe/fachinformatiker-anwendungsentwicklung/gehalt/
Recherche nach Tarifentgelt für ein Auszubildenden Fachinformatiker für Anwendungsentwicklung im dritten Lehrjahr

https://www.php.net/
https://www.php.net/manual/en/class.dateinterval.php
https://www.php.net/manual/de/function.class-exists.php
https://www.php.net/manual/de/function.method-exists.php
https://www.php.net/manual/en/language.oop5.cloning.php
Sprachenreferenz für PHP

https://developer.mozilla.org/en-US/
https://www.w3schools.com/
Sprachreferenz für weitere Webtechnologien

http://help.innovator.de/11.5/de_de/InnovatorX/Content/Ref.MetaM.UML/CLComponentDia.htm
Recherche Komponentendiagramme

https://www.ionos.de/digitalguide/websites/web-entwicklung/sequenzdiagramme-mit-uml-erstellen/
Recherche Sequenzdiagramm

https://www.iot-design.de/allgemein/software-architektur-designdie-definition-wie-alles-ineinander-wirkt/
Architekturdesign Recherche

https://stackoverflow.com/questions/28139621/shortcut-for-denoting-or-implying-getters-and-setters-in-uml-class-diagrams
Get/Set Notation vom Beitrag des Autors Gerd Wagner vom 31 Januar 2017 um 19:55 Uhr übernommen – siehe dazu Notiz aus dem Klassendiagramm

https://www.philipphauer.de/study/se/design-pattern/singleton.php#gof
Singleton Entwurfsmuster Beschreibung

https://www.wintotal.de/thin-client/
Informationen und Vor/Nachteile von Thin-Clients

IT-Handbuch für Fachinformatiker
8., aktualisierte Auflage 2017
ISBN: 978-3-8362-5466-3
Autor: Sascha Kersken
Rheinwerk Computing Verlag

S. 851, Cloud Computing
S. 202, Webserver
S. 1037, Webserveranwendungen

IT-Berufe
Entwickeln und Bereitstellen von Anwendungssystemen
ISBN: 978-3-14-225384-8
Autoren: K. Ringhand / H.-G- Wittmann
Westermann Verlag

S. 332, 8.4.3.1 Äquivalenzklassenbildung
S. 333, 8.4.3.2 Grenzwertanalyse