Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Programmiersprachen

Entwicklung eines Wochenplaners für Studierende, im Umfeld des Modulverwaltungssystems der Freien Universität Berlin

Martin Schlegel
Matrikelnummer: 4666893
legelhcs@fu-berlin.de

Betreuerin und Erstgutachterin: Prof. Dr. Margarita Esponda-Argüero
Zweitgutachter: Prof. Dr. Raúl Rojas

Berlin, 1. April 2016

Zusammenfassung

Das Ziel der vorliegenden Bachelorarbeit war die Entwicklung einer Webanwendung, die zum Erstellen eines Stundenplans dienen soll. Dabei sollte es möglich sein einen Stundenplan automatisch berechnen zu lassen, wobei der Benutzer die Berechnung an seine Bedürfnisse anpassen können soll. Zusätzlich sollte die Software in die bestehende Webseite des Modulverwaltungssystems der Freien Universität Berlin integriert werden. Diese Arbeit führt durch die verschiedenen Schritte des Projektes und geht speziell auf die technische Umsetzung und die dabei aufkommenden Probleme und Lösungen ein. Schlussendlich werden weitere Verbesserungsmöglichkeiten vorgestellt, welche innerhalb des Projektes nicht mehr umgesetzt werden konnten.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

1. April 2016

Martin Schlegel

Inhaltsverzeichnis

1 Einführung

Dieses Kapitel soll die Motivation, die Aufgabenstellung, die Ziele und den Aufbau der Bachelorarbeit verdeutlichen.

1.1 Motivation

Wichtig für mich war es ein Problem nicht nur theoretisch zu betrachten, sondern eine Bachelorarbeit mit praktischem Bezug zu bearbeiten und sich somit auch mit der konkreten Lösung zu beschäftigen. Innerhalb meines Studiums habe ich in den ersten Wochen eines jeden Semesters viel Zeit damit verbringen müssen einen für mich passenden Stundenplan zu erstellen, der letztendlich nicht perfekt auf meine Bedürfnisse angepasst war. Somit gab es die Idee, aufbauend auf der Arbeit von Christoph Graebnitz, dieser erstellte einen Studienverlaufplaner[1] für Studenten, einen Wochenplaner für Studenten zu entwickeln, der genau diese Aufgabe übernimmt und somit den Einsteig ins Studium vereinfacht.

1.2 Aufgabenstellung und Ziele

Am Ende dieser Arbeit soll eine Webanwendung zur Verfügung stehen, welche es den Studierenden ermöglichen soll ihr Semester konkret planen zu lassen. Es soll eine fachbereichsübergreifende Lösung entstehen und nicht nur speziell an den Fachbereich Informatik angepasst werden. Dabei ist es wichtig, dass keine Anmeldung zur Benutzung der Software notwendig ist, da auch Studenten, welche sich vor dem ersten Semester befinden und Studieninteressenten darauf zugreifen können sollen. Daraus resultiert auch die Anforderung nach einer Exportmöglichkeit des fertigen Stundenplans. Zusätzlich soll sich die Anwendung nahtlos in das bestehende Bedien- und Grafikkonzept des MVS[2] einbinden. Dieses stellt ein System für die Kurz- und Langzeitplanung von Veranstaltungen für Studenten und Dozenten innerhalb verschiedener Fachbereiche der Freien Universität dar.

Die Zielsetzung der Abschlussarbeit gliedert sich in projektbezogene und persönliche Ziele auf. Folgende Anforderungen sind in der Bearbeitung der Projektaufgabe zu erfüllen um am Ende als erfolgreich und nutzbringend bewertet zu werden.

- Eine in das MVS integrierte Webanwendung, die es den Studierenden ermöglicht aufbauend auf ihren Studienverlaufsplan das konkrete Semester zu planen. Dabei ist es wichtig, dass alle auch vom MVS integrierten Fachbereiche unterstützt werden.

[1] Der Studienverlaufplaner bietet dem Benutzer die Möglichkeit das Studium über mehrere Semester hinweg zu planen.

[2] Modulverwaltungssystem

- Die Kernfunktion des Wochenverlaufplaners soll es sein, dem Benutzer einen für ihn passenden Stundenplan zu berechnen.
- Die Darstellung soll veranstaltungsorientiert erfolgen.
- Der Stundenplan muss einstündige Veranstaltungen darstellen können.
- Der Benutzer muss den Stundenplan exportieren können.
- Die grafische Oberfläche soll klar strukturiert, leicht verständlich sein und sich somit in die bestehende Umgebung einbinden.

Zum anderen verfolgt die Arbeit die Weiterbildung der persönlichen Fähigkeiten im Bereich der Webentwicklung.

- Der Ausbau der Fähigkeiten in Bereich der Frontendentwicklung, welches in diesem Projekt konkret mittels HTML[3], CSS[4], Bootstrap, JavaScript, jQuery und Thymeleaf umgesetzt wird.
- Das Erlangen von neuem Wissen in der Backendentwicklung. Dies beschränkt sich in diesem Projekt auf das Spring Framework in Java.

1.3 Aufbau der Arbeit

Das erste Kapitel dient der Einführung in das Themengebiet der Arbeit. Die Grundlagen, welche zur Umsetzung der Anwendung nötig sind werden im zweiten Kapitel erläutert. Im dritten Abschnitt werden die Analyse der Anforderungen und die daraus resultierende Ablaufplanung vorgestellt. Kapitel Vier gibt einen ausführlichen Überblick über die Implementierung der Anwendung mit anschließender Fehleranalyse. Im letzten Kapitel werden die Ergebnisse zusammengefasst und ein Ausblick gegeben, welche Erweiterungen die Anwendung komplettieren würden.

[3] Hypertext Markup Language
[4] Cascading Style Sheets

2 Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen zum Thema der Bachelorarbeit. Da das Hauptaugenmerk dieser Arbeit auf den Schedulingalgorithmus liegt und dieser ein Optimierungsalgorithmus darstellt, kommt im Folgenden eine allgemeine Einführung zu Optimierungsproblemen. Dabei werden die Schwierigkeiten beim Lösen dieser Probleme und verschiedene Lösungsmöglichkeiten betrachtet.

2.1 Optimierungsprobleme

Ein Optimierungsproblem wird durch einen gegebenen Lösungsraum sowie durch eine Bewertungsfunktion definiert. Das Ziel ist Aussagen über die verschiedenen Werte der Lösungen zu treffen oder einen möglichst mini- oder maximalen Wert zu finden. Ist dies der Fall spricht man von einem Mini- oder Maximierungsproblem. Die Optimierungsprobleme lassen sich in kombinatorische Optimierungsprobleme und lineare Optimierungsprobleme unterteilen. Dabei ist das Ziel bei einem kombinatorischen Optimierungsproblem aus einer gegebenen diskreten Menge eine Teilmenge zu konstruieren, welche unter Umständen gewissen Nebenbedingungen erfüllt und bezüglich einer Gewichtsfunktion optimal ist. Lineare Optimierungsprobleme beschäftigen sich mit der Optimierung linearer Zielfunktionen und einer Menge, die durch lineare Gleichungen und/ oder Ungleichungen eingeschränkt ist. Des Weiteren ist eine Einteilung in die folgenden drei Unterpunkte möglich.

Entscheidungsproblem Hierbei wird ein zusätzlicher Grenzwert gegeben, zu dem ermittelt werden soll ob es eine Lösung gibt, dessen Wert den gegebenen Grenzwert über- oder unterschreitet.

Suchproblem Ein Suchproblem hat das Ziel eine konkrete Lösung zu finden, welche einer gegebenen Mindestanforderung erfüllt.

Optimierungsproblem Das direkte Optimierungsproblem ist sozusagen ein spezielles Suchproblem, bei dem die Lösung mit dem insgesamt besten Wert gesucht ist.

Ein Algorithmus, der ein Optimierungsproblem löst, nennt man Optimierungsalgorithmus, löst ein Algorithmus ein Optimierungsproblem hingegen nur näherungsweise bezeichnet man ihn als Approximationsalgorithmus.

2.2 Komplexität der Stundenplansuche

Die Suche nach einem optimalen Stundenplan ist ein kombinatorisches direktes Optimierungsproblem, da aus der Menge der gegebenen möglichen Termine eine Untermenge gesucht werden soll, die durch eine gegebene Gewichtsfunktion optimal sein soll. Dabei sind im speziellen Anwendungsfall

zusätzlich verschiedene Nebenbedingungen zu beachten. Angenommen a ist die Anzahl der zu betrachtenden Kurse und b ist die maximale Anzahl verschiedener Terminmöglichkeiten pro Kurs, dann drückt b^a die maximale Anzahl an verschiedenen berechenbaren Stundenplänen aus. Mittels einem Brute-Force-Ansatz, der alle Möglichkeiten berechnet und bewertet, ist das hier zu betrachtende Problem nicht in Polynomialzeit lösbar.

2.3 Theoretische Lösungsmöglichkeiten

Dieser Abschnitt gibt einen kurzen Einblick wie Optimierungsprobleme in der Theorie gelöst werden können.

2.3.1 Exakte Lösungsmöglichkeiten

Die folgenden Algorithmen stellen nur eine Auswahl verschiedenster Möglichkeiten dar, um das Problem exakt zu Lösen.

Systematisches Durchsuchen Mittels diesem Verfahren werden alle Permutationen generiert, bewertet und die bisher beste Lösung gespeichert, auch Brute-Force-Ansatz genannt.

Dynamische Programmierung Dieses Verfahren ist nur dann anwendbar, wenn sich das Problem in mehrere gleichartige Teilprobleme aufteilen lässt und sich die optimale Lösung des Problems aus den optimalen Lösungen der Teilprobleme zusammensetzt.

Branch-and-Bound Verfahren Hierbei wird das Problem wiederum in verschiedene Teilprobleme aufgespalten (Branch). Anschließen ist es möglich suboptimale Lösungen frühzeitig durch geeignete Schranken (Bound) zu erkennen und auszuschließen. Dies hat zur Folge, dass der zu durchsuchende Lösungsraum möglichst klein gehalten wird und sich somit die Rechenzeit verringert.

2.3.2 Approximationsalgorithmen

Die in diesem Unterkapitel vorgestellten Heuristiken stellen nur eine Auswahl dar und soll nur einen kleinen Einblick in die Berechnungsvielfalt dieser Probleme geben. Bei diesen Lösungsverfahren ist zu beachten, dass es nicht bestimmbar ist, ob man ein optimales Resultat erreicht hat, beziehungsweise wie weit man von einem Optimum entfernt ist. Der Ansatz aller Heuristiken ist dabei ähnlich. Zunächst wird eine Lösung gewählt, zum Beispiel mittels einem greedy[5] Verfahren. Anschließend wird diese Lösung innerhalb einer

[5]Dies ist ein heuristisches Optimierungsverfahren, welches sich in jedem Schritt für die momentan erfolgversprechendste Alternative entscheidet. Quelle: [wirhr]

Schleife nach und nach verbessert und somit eine optimaleres Ergebnis erzielt. Die Abbruchbedingung der Schleife kann dabei unter anderem von bestimmten Schranken oder von der Anzahl der Durchläufe abhängen. Bekannte Aproximationsalgorithmen sind die Tabu-Suche, der Hinauf-Algorithmus oder der Treshold-Accepting-Algorithmus.

2.4 Vorhandene Lösungen

Natürlich gibt es auch die Möglichkeit vorhandene Lösungen zu benutzen. Dabei können wiederum verschiedene Probleme auftreten. Zum einen existieren nicht viele frei verfügbare Frameworks und zum anderen sind diese meist für komplexere Planungen vorgesehen. Diese Algorithmen beachten zum Beispiel zusätzliche Ressourcen wie Räume und Lehrkräfte, welche bei unserem Problem nicht beachtet werden sollen. Zusätzlich gibt es kein Framework in JavaScript oder PHP[6], was nötig ist, da der Wochenverlaufplaner clientbasiert arbeiten soll. Dies alles hat zur Folge, dass viel Anpassungsarbeit notwendig sein und dadurch die Zeitersparnis gering ausfallen wird. Somit ist es besser den Algorithmus neu umzusetzen um ihn exakt an die gegebenen Anforderungen anpassen zu können. Eine Anforderung an den Algorithmus ist, dass die gefundene Lösung eine optimale Lösung darstellt. Des Weiteren kann man die Annahme treffen, dass ein realer Student pro Semester nicht mehr als 7 verschiedene Kurse belegt und ein Kurs im Durchschnitt maximal 10 verschiedene Terminmöglichkeiten anbietet. Nimmt man diese Werte als oberste Schranke an gibt es 10^7 also maximal 10 Millionen Möglichkeiten für den resultierenden Stundenplan. Heutige Computer sind in der Lage diese Möglichkeiten in einer annehmbaren Zeit zu berechnen. Da nur das Probieren aller Möglichkeiten eine optimale Lösung garantiert und die Tatsache, dass der Algorithmus in der Praxis die Berechenbarkeit nicht übersteigt führte dazu, dass eine exakte Lösungsmöglichkeit im folgenden Projekt umgesetzt wird. Weil das Problem sich nicht gut in Teilprobleme aufteilen lässt und brauchbare Schranken schwierig zu definieren sind, wird das systematische Durchsuchen angewandt.

[6] Hypertext Preprocessor

3 Anforderungsanalyse und Ablaufplanung

In diesem Kapitel werden zunächst die an die Anwendung gestellten Anforderungen analysiert. Anschließen wird das daraus resultierende Umsetzungskonzept dargestellt.

3.1 Anforderungsanalyse

Wenn man eine Anwendung in eine schon bestehende Umgebung integrieren möchte ist es unabdingbar, dass man neben dem nötigen technischen Fachwissen zusätzlich auch über die Besonder- und Einzelheiten des MVS informiert ist. Um im Allgemeinen eine moderne Webseite zu schreiben ist es nötig Grundlagenwissen in den Bereichen HTML, JavaScript und CSS zu besitzen. Dabei bildet HTML, als textbasierte Auszeichnungssprache, die statische Komponente der Webanwendung, wohingegen JavaScript, als eine Skriptsprache, für die dynamische Veränderung der angezeigten Inhalte benutzt wird. Zusätzlich wird mittels CSS, als Stylesheet-Sprache, getrennt vom Inhalt der Webanwendung die Gestaltung übernommen. In der Umgebung des MVS werden zusätzliche Frameworks eingesetzt, mit dem Ziel das implementieren von Komponenten zu vereinfachen. Dazu gehören jQuery, welches eine JavaScript Bibliothek darstellt und zusätzliche Funktionalitäten bereitstellt. Unter anderem zum Beispiel asynchrone HTTP[7]-Anfragen mittels AJAX[8] oder ein erweitertes Event-System um auf Benutzereingaben besser reagieren zu können. Ein weiteres Framework stellt Bootstrap dar, welches auf HTML und CSS basiert. Es beinhaltet einheitliche Gestaltungsvorlagen, welche ein modernes und übersichtliches Userinterface ermöglicht. Zusätzlich wird die Webanwendung automatisch an verschiedene Bildschirmgrößen angepasst. Da die Webanwendung auch von internationalen Studenten benutzt wird, ist diese auch in Englisch verfügbar. Um die Lokalisierung von Texten zu bewerkstelligen wird Thymeleaf verwendet. Alle vorangegangenen Technologien gehören zur clientseitigen Architektur. Auf der Seite der serverseitigen Architektur wird im MVS mit Java und im speziellen mit dem Spring-MVC[9]-Framework gearbeitet. Da innerhalb des MVS schon viele Service zum Zugriff auf die im Hintergrund liegende Datenbank zur Verfügung stehen, ist ein detailliertes Wissen mit dem Umgang der Datenbank nicht essentiell.
Auf Grund der Tatsache, dass die anzuzeigenden Daten teilweise aus dem Studienverlaufplaner stammen, muss dieser um benötigte Funktionalitäten erweitert werden. Daraus schlussfolgert sich, dass genaues Wissen über den Aufbau der Anwendung notwendig und somit eine tiefe Einarbeitung nötig ist. Genau wie der Studienverlaufplaner soll auch der Wochenverlaufplaner

[7] Hypertext Transfer Protocol
[8] Asynchronous JavaScript and XML
[9] Modell-View-Controller

ohne Anmeldung in das MVS voll funktionsfähig sein. Daraus resultiert auch hier eine stark clientseitige Architektur, da der Server keinerlei Wissen über den momentanen Benutzer besitzt.
Die übersichtliche Darstellung der verschiedenen Komponenten der Anwendung, beispielsweise der Stundenplan und verschiedene Buttons zur Umsetzung der verschiedenen Funktionen ist relativ komplex. Daraus resultiert eine ständige Rücksprache mit anderen Studenten, um zu verhindern, dass das Designkonzept in eine falsche Richtung läuft und die Webanwendung letztendlich unbrauchbar erscheinen lässt. In den letzten Jahren ist der Erfolg von Software immer abhängiger von der Präsentation und der Nutzeroberfläche geworden. Webanwendungen bilden hierbei keine Ausnahme.

3.2 Ablaufplanung

Um sicherzustellen, dass ein Projekt dieser Größe erfolgreich und mit so wenig zusätzlichen Überarbeitungsaufwand zu Ende gestellt werden kann, ist es sehr hilfreich einen detaillierten Ablaufplan der Teilschritte zu erstellen. Dieser führt chronologisch durch das gesamte Projekt und soll die Zwischenziele verdeutlichen. Ein Problem hierbei ist, dass ein Fehler oder eine Fehlentscheidung in einem frühen Schritt des Projektes sich unter Umständen sehr stark auf die nachfolgenden Schritte auswirken und eine Fehlerkette entstehen kann, dessen Lösung viel Zeit in Anspruch nimmt. Aus diesem Grund ist die Vermeidung von Fehlentscheidungen im Planungsprozess, zum Beispiel durch eine detaillierte Anforderungsanalyse, sehr wichtig.

1. Einarbeitung in die technischen Grundlagen
    - HTML
    - CSS
    - JavaScript
    - JQuery
    - Bootstrap
    - Thymeleaf
    - Spring Framework (im speziellen das MVC-Framework)
2. Einarbeitung in das bestehende MVS
    - Modulsystem und das dazugehörige Datenbankschema verstehen
    - Einarbeitung in den Studienverlaufplaner
3. Erstellen einer Funtionalitätenliste
4. Planung des Designs
    - Stundenplan

- Bedienelemente
- Fehleranzeige

5. Implementierung der Webanwendung
    - Umsetzung des Designs
    - Ausarbeitung der Architektur
    - Umsetzung der Funktionalitätenliste
    - Fehlerbereinigung

4 Durchführung des Projektes

Der folgende Abschnitt stellt die Umsetzung der einzelnen Planungsschritte im Detail dar. Dabei wird im speziellen auf auftretende Probleme sowie deren Lösung und die Gründe für wichtige Entscheidungen beziehungsweise Vorgehensweisen eingegangen.

4.1 Einarbeitung in die technischen Grundlagen

Um eine Webanwendung zu implementieren ist das Wissen über die verwendeten Technologien essentiell. Im Falle von HTML, CSS, JavaScript, jQuery und Bootstrap wurde das Wissen vorwiegend von [w3shr] bezogen. Wohingegen Thymeleaf sehr gut auf [thyhr] und das Spring MVC-Framework auf [sprhr] dokumentiert ist. Während der Bearbeitung von anderen Projekten habe ich festgestellt, dass die effektivste und schnellste Möglichkeit den Umgang mit neuen Technologien zu erlangen darin bestehlt praktisch mit dieser zu arbeiten. Deshalb war es unabdingbar zu einem sehr frühen Zeitpunkt des Projektes mit der Implementierung der Anwendung anzufangen, auch wenn in dieser Phase noch relative viele Fehler gemacht werden. Jedoch ist der Lerneffekt durch das Verursachen und Erkennen von Fehlern und deren selbständige Lösung am größten.

4.2 Einarbeitung in das bestehende Modulverwaltungssystem

Nach dem Erlernen der benötigten Grundlagen war es notwendig, den bestehenden Code einer Durchsicht zu unterziehen. Im speziellen Fall des Studienverlaufplaners, da diese Anwendung direkt daran aufbaut, war eine detaillierte Einarbeitung in den Code notwendig um diesen in den folgenden Schritten ergänzen zu können. Aufbauend auf der Durchsicht war es relativ zeitnah möglich einen eigenen Controller und View für den Wochenverlaufplaner zu implementieren. Um diesem View durch den Controller die anzuzeigenden Daten zu übergeben war es zunächst Erforderlich mehr über die Datenbank zu erfahren. Dies erfolgte durch eine Einweisung von Frau Prof. Dr. Margarita Esponda-Argüero und Stephan Sundermann[10]. In der Abbildung 1 auf Seite 10 ist der benötigte Datenbankauschnitt als ER[11]-Modell zu sehen. Der Ausgangspunkt sind die Events (*CourseIDs*), welche im Studienverlaufplaner ausgewählt werden. Wichtig hierbei ist zu erwähnen, dass jeder *Course* einem bestimmten Modul zugeordnet ist. Die benötigten Daten sind die einzelnen *Appointments*. Ausgehend von den *Coursen* besitzten diese mehrere *AncillaryCourse*. Zu einem *Course* und *AncillaryCourse* gehören, falls die Daten noch nicht terminiert sind, vorübergehende Serien

[10] Kernentwickler des MVS
[11] Entity-Relationship

(*TentativeSerie*). Sind die Daten schon terminiert, existieren normale *Serien*. Vorübergehende und normale *Serien* beinhalten *Appointments*, welche sich im Inhalt ihrer gespeicherten Daten unterscheiden. Je nach Zeitpunkt der Anfrage können demzufolge mehr oder weniger genaue Daten zu einem *Course* angezeigt werden.

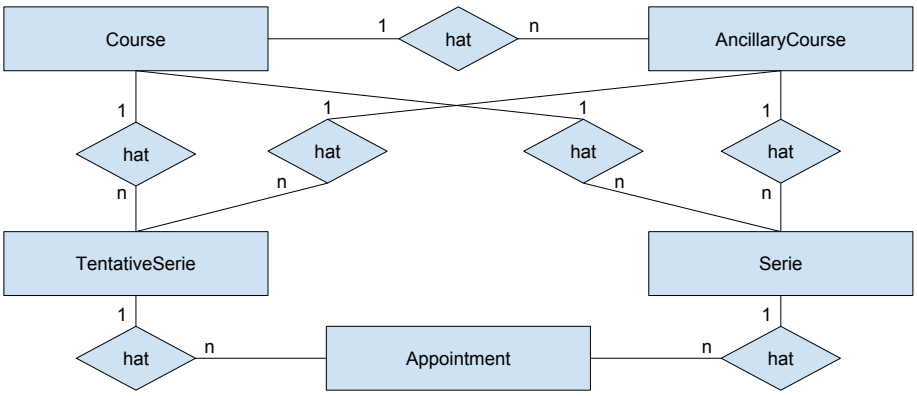

Abbildung 1: Abstrahierte Darstellung des von mir verwendeten Ausschnitts des Modulsystems. Notiert in Chen-Notation gelesen von links nach rechts und von oben nach unten.

4.3 Erstellen einer Funktionalitätenliste

Das folgende Kapitel beschäftigt sich mit der Entwicklung einer Funktionalitätenliste, welche zusammen genommen den Wochenverlaufplaner bilden. Die Reihenfolge entspricht dabei der chronologischen Abarbeitung innerhalb des Projektes. Der Prozess verlief in einem iterativen Ablauf, da es mehrmals zu nachträglichen Verbesserungsideen kam. Somit wurde die Liste im Verlauf des Projektes mehrmals um verschiedene Punkte ergänzt oder die detailierte Umsetzung einzelner Anforderungspunkte verändert. Dies ist die finale Funktionalitätenliste, wie sie am Ende des Projektes umgesetzt wurde. Ergänzend dazu ist in Abbildung 2 auf Seite 12 der Workflow der Anwendung dargestellt.

1. Erweiterung des Studienverlaufplaners:
   Da der Wochenverlaufplaner auf den Studienverlaufplaner aufbauen sollte, ist es zunächst nötig dessen Funktionalität zu erweitern. Dazu gehört die lokale Speicherung der ausgewählten *CourseIDs* auf die auch vom Wochenverlaufplaner zugegriffen werden muss und das implementieren eines Buttons, der den Benutzer zur neuen Anwendung weiterleitet.

2. Laden der *Course*-Daten:
   In diesem Schritt müssen die zu den *CourseIDs* zugehörigen Daten vom Backend geladen werden. Der Großteil der zu implementierenden Funktion befindet sich im Backend, da die umzusetzende Funktionalität im Frontend sich darauf beschränkt die Anfrage an das Backend zu stellen und die erforderten Daten lokal abzuspeichern.

3. Visualisierung der Veranstaltungen:
   Nachdem die Daten vorhanden sind müssen diese angezeigt werden. Dies ist somit der erste Punkt, welcher eine designtechnische Umsetzung erfordert.

4. Manuelle Aktivierung der Termine:
   Zusätzlich zum Schedulingalgorithmus soll es dem Benutzer auch möglich sein seinen Stundenplan manuell zu planen.

5. Visualisierung zusammengehörender Veranstaltungen:
   Um die Übersichtlichkeit zu erhöhen sollen zusammengehörende Veranstaltungen mit der selben Farbe eingefärbt werden. Da die Anzahl der ausgewählten Veranstaltungen nicht klar definierbar ist, ist eine dynamische Lösung erforderlich.

6. Anzeige von nicht eintragbaren Veranstaltungen und Terminen:
   Veranstaltungen, welche fehlerhafte Daten besitzen, müssen gesondert angezeigt werden. Zusätzlich werden Termine am Wochenende, vor 8 Uhr und nach 20 Uhr auch seperat angezeigt. Dies hat den Grund, dass die Anzeige dieser Termine die gesamte Darstellung sehr unübersichtlich macht und Termine zu diesen Zeiten sehr selten vorkommen.

7. Entwicklung des Schedulingalgorithmus:
   Die automatische Suche nach einem passenden Stundenplan für den Benutzer stellt die Kernfunktion des Wochenverlaufplaners dar.

8. Aus- und Einblenden von nicht aktivierten Terminen:
   Die Anzeige von nicht aktivierten Terminen schränkt die Übersichtlichkeit des Stundenplans stark ein, deshalb soll es möglich sein diese Auszublenden.

9. Entfernung einzelner Veranstaltungen:
   Die Auswahl der angezeigten und im Algorithmus beachteten Veranstaltungen soll auch innerhalb des Wochenverlaufplaners nachträglich veränderbar sein.

10. Dynamisierung des Schedulingalgorithmus:
    Um die Ergebnisse des Schedulingalgorithmus besser an den jeweiligen Benutzer anzupassen, soll es diesem möglich sein bevorzugte Tage und Zeiten anzugeben.

11. Export des Stundenplans:
    Der Stundenplan muss in eine entsprechende Kalenderdatei exportiert werden können, da es dem Benutzer nicht möglich ist sich lokal anzumelden und seinen Stundenplan zu speichern.

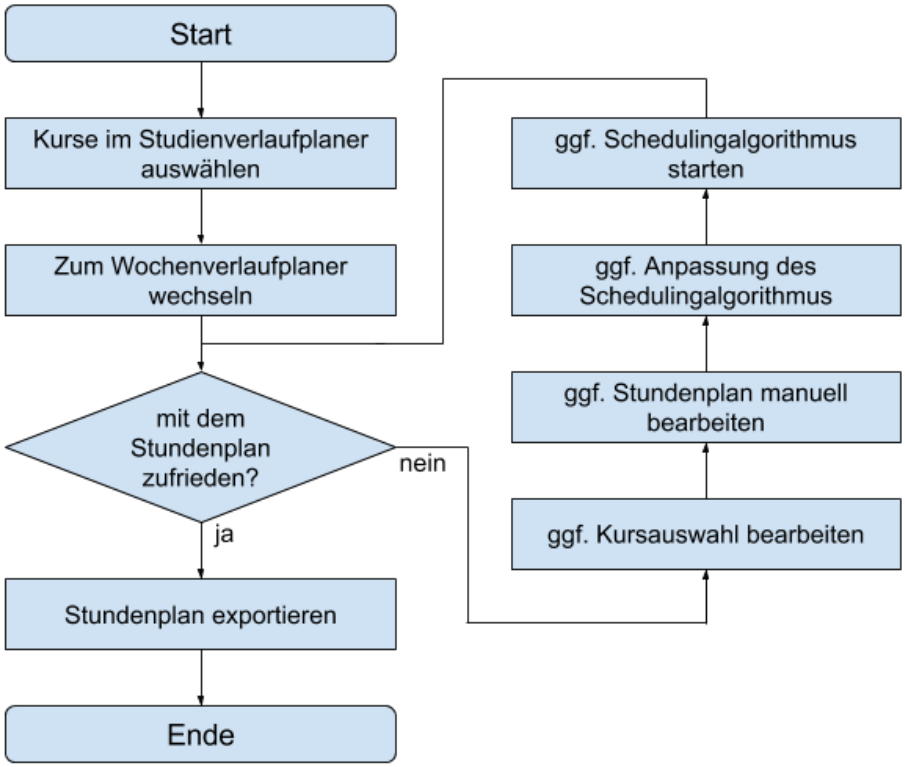

Abbildung 2: Workflow des Wochenverlaufplaners

4.4 Planung und Entwicklung des Designs

Aufbauend auf den ermittelten Funktionen ist es wichtig ein darauf passendes Designkonzept zu entwickeln. Das Design muss übersichtlich und einfach sein, hinzu kommt, dass es sich in das bestehende Design des MVS integrieren soll um die Benutzbarkeit innerhalb der gesamten Anwendung zu gewährleisten. Zusätzlich soll das Design die verfügbaren Funktionalitäten verdeutlichen und die Anwendung somit sich selbst erklärend darstellen. Da die Anwendung für Studenten, welche größtenteils im Alter von 17 bis 30 Jahren alt sind, benutzt wird, kann man davon ausgehen, dass die Nutzergruppe weitreichende Erfahrungen mit dem Umgang von Webanwendungen besitzt. Um die genauen Anforderungen der Studenten zu ermitteln war zunächst geplant eine Umfrage durchzuführen. Aufgrund von Zeitmangel war

dies im Rahmen der Arbeit nicht mehr umsetzbar und die Anforderungen mussten größtenteils aus meiner persönlichen Perspektive erstellt werden. Das letztendliche Design ist in Abbildung 3 auf Seite 13 einsehbar. Im Vergleich dazu ist in Abbildung 4 auf Seite 14 das bestehende Design des Studienverlaufplaners zu sehen.

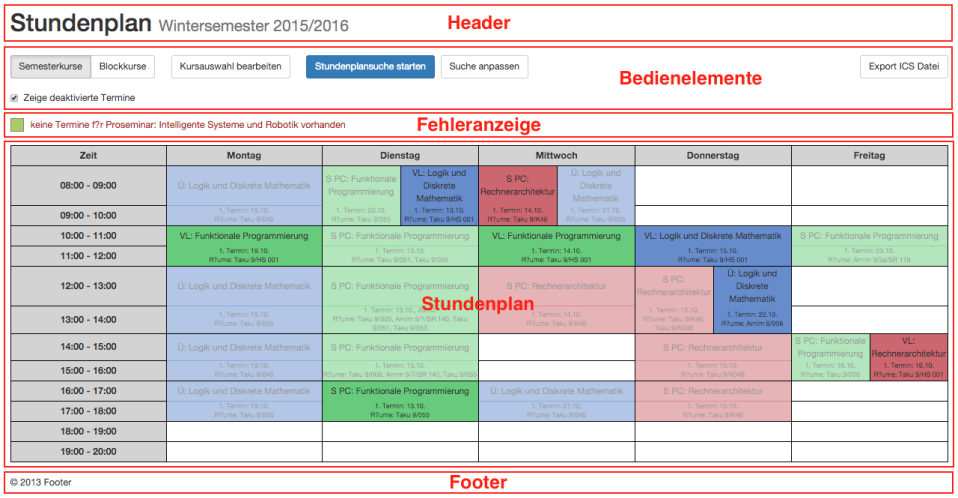

Abbildung 3: Gliederung der Benutzeroberfläche

4.4.1 Stundenplan

Im Zentrum der Anwendung befindet sich die Anzeige des Stundenplans. Damit dieser sich in das Umfeld des MVS integriert wird die standartisierte Tabelle von HTML und Bootstrap benutzt, wie sie auch schon in anderen Teilen des MVS Anwendung findet. Die Datenbank des MVS bietet viele verschiedene Informationen zu einem *Appointment* an, wobei bei der Anzeige in einem Stundenplan nur wenige Daten relevant sind. Dazu gehören Tag und Uhrzeit, der Name des dargestellten *Courses*, das Datum des ersten Termins und die Räume. Die Darstellung der Tag- und Uhrzeitinformationen werden dabei durch die Eintragung in den entsprechenden Zellen der Tabelle realisiert. Kommt es zu dem Fall, dass mehrere *Appointments* in der selben Zelle stattfinden, werden diese nebeneinander angezeigt. Bei der Anzeige von sehr vielen *Appointments* innerhalb einer Zelle wird die Darstellung sehr unübersichtlich. Dieser Sonderfall tritt aber nur sehr selten ein und eine andere Darstellungsweise, wie das Platzieren untereinander würde die Anzeige noch weiter verkomplizieren. Innerhalb eines *Appointment* wird der abgekürzte Name als Hauptinformation gefolgt von der Datums- und Rauminformation angezeigt. Bewegt sich die Maus über ein *Appointment* wird der gesamte Name angezeigt. Um zusammengehörende *Appointments*

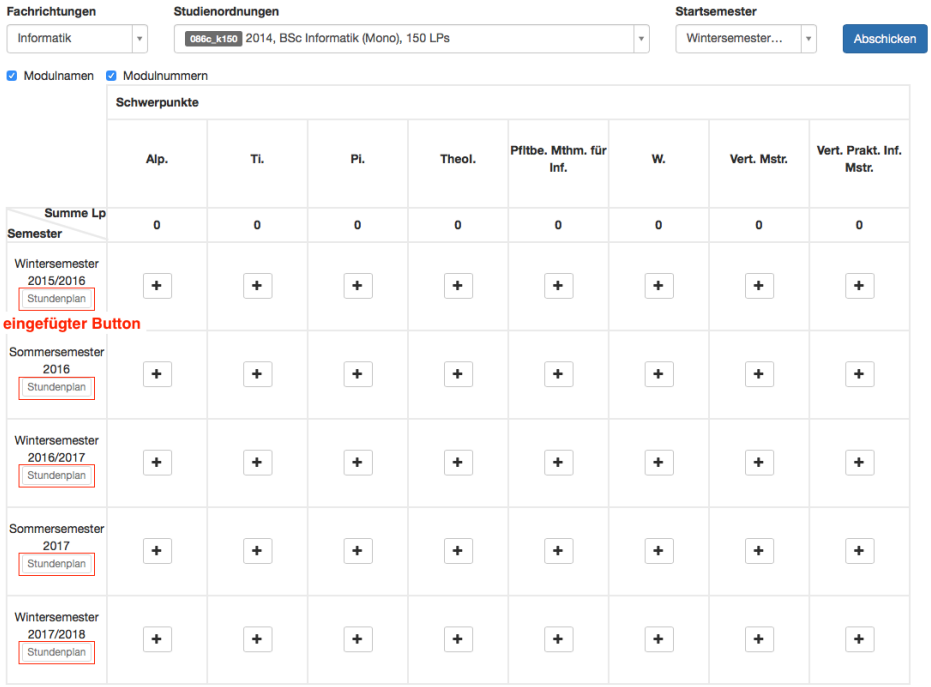

Abbildung 4: Interface des Studienverlaufplaners

auch visuell darzustellen, werden deren Hintergründe gleich eingefärbt, dabei ist es wichtig nur Farben zu benutzen, welche einen hohen Kontrast zu der Schriftfarbe besitzen um diese weiterhin lesen zu können. *Appointments*, welche momentan nicht aktiviert sind, werden dadurch visualisiert, dass deren Hintergrundfarbe einen Alphawert von 0,5 bekommt. Um die Möglichkeit der manuellen Aktivierung eines *Appointments* zu verdeutlichen, wird zusätzlich zur Alphawertänderung der Mauszeiger bei allen nicht aktivierten *Appointments* als Pointer dargestellt.

4.4.2 Bedienelemente

Um dem bestehenden Design zu folgen werden alle Bedienelemente, wie zum Beispiel beim Studienverlaufplaner, direkt unter dem Header angezeigt. Dies ermöglicht dem Benutzer alle zur Verfügung stehenden Funktionen zu überblicken. Dabei wird der Button zum Start des Schedulingalgorithmus gesondert dargestellt um dessen Kernfunktionalität zu unterstreichen.
Das Bearbeiten der Kursauswahl erfolgt über ein eingeblendetes Modal, Abbildung 5 auf Seite 15. In diesem wird eine nach *ModuleID* sortierte Liste dargestellt, an der es möglich ist sofort den Auswahlstatus jedes einzelnen *Courses* und *AncillaryCourses* zu erkennen und zu ändern.
Die Dynamisierung des Schedulingalgorithmus wird auch mittels einem Mo-

dal realisiert, Abbildung 6 auf Seite 16. Die Funktionsweise und Darstellung ist dabei der Webseite Doodle[12] nachempfunden. Mittels dieser Webseite ist es einer Gruppe möglich den für sie besten Termin zu vereinbaren, indem jeder Teilnehmer die Termine individuell priorisieren kann. Das entspricht nahezu der gewünschten Funktionalität im Wochenverlaufplaner. Dabei ist noch positiv zu erwähnen, dass Doodle ein weit verbreitetes Tool im Bereich Terminfindung darstellt und die Anlehnung daran einen positiv wirkenden Wiedererkennungseffekt hervorruft. Es gibt drei Prioritätsstufen, welche durch Ja, (Ja), und Nein, sowie durch die Farben Grün, Gelb und Rot dargestellt werden. Diese lassen sich auf einen gesamten Zeitslot, auf einen ganzen Tag und/ oder individuell pro Zelle auswählen. Durch die kompakte und übersichtliche Darstellung ist es dem Nutzer möglich innerhalb kurzer Zeit die bekannte Funktionsweise nachzuempfinden und mit wenigen Klicks die Suche seinen Bedürfnissen anzupassen.
Die Möglichkeit nicht aktivierte *Appointments* aus- und einzublenden ist mittels einer Checkbox dargestellt, da dadurch auch die aktuelle Einstellung darstellbar ist.

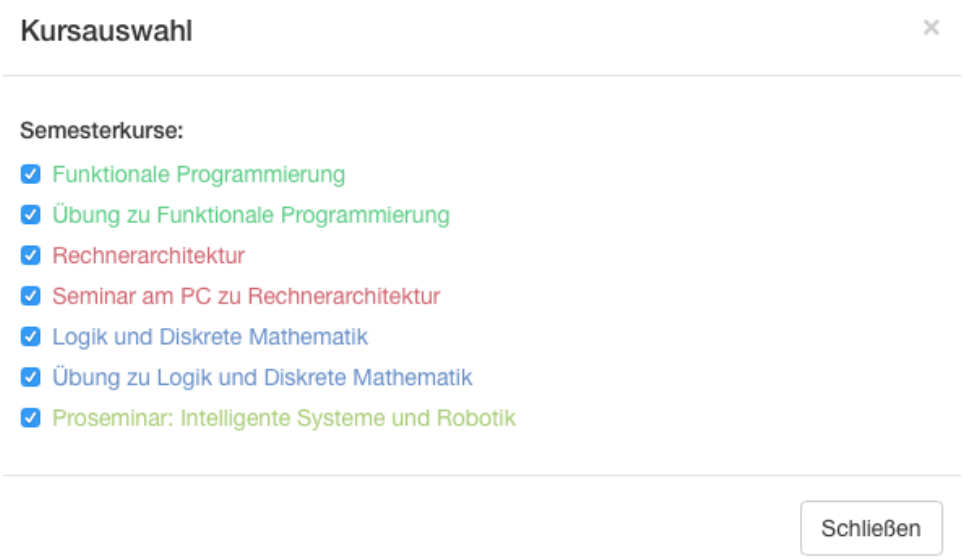

Abbildung 5: Kursauswahlmodul

4.4.3 Fehleranzeige

Die Anzeige von Fehlern wird mit roter Schrift zentral zwischen den Bedienelementen und der Stundenplananzeige vorgenommen um den Benutzer sofort darüber in Kenntnis zusetzten. Ein Anwendungsfall hierfür ist zum

[12] www.doodle.com, 07.01.2016 19.55 Uhr

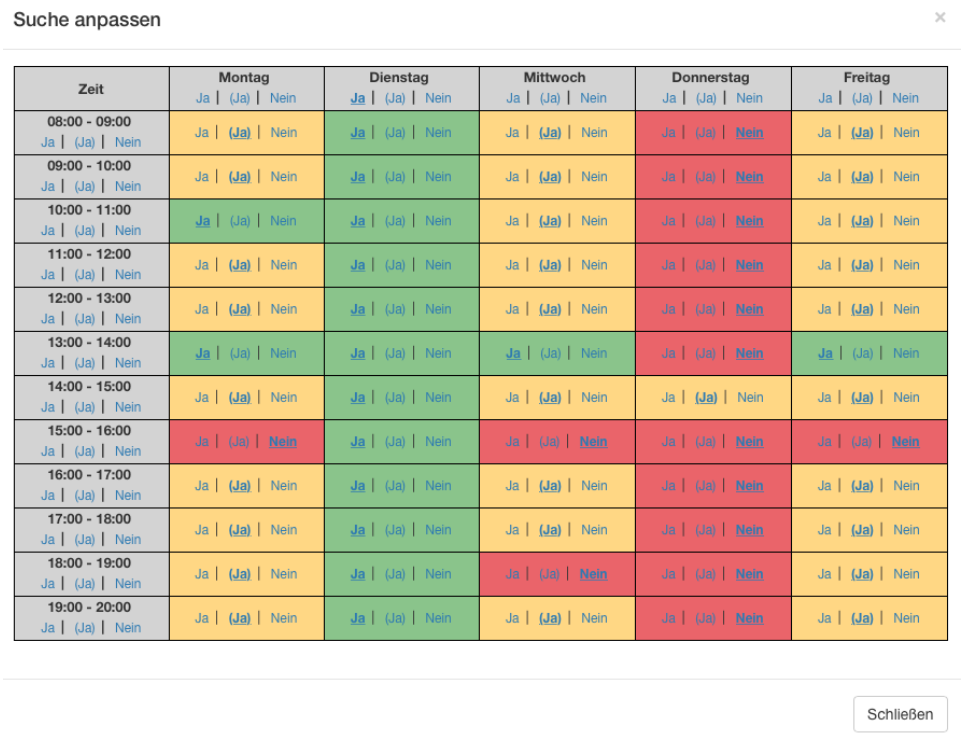

Abbildung 6: Anzeige der Schedulingalgorithm-Anpassung als Modul

Beispiel das Fehlen von *Appointments* zu einem *Course*. Dem Benutzer wird sofort vermittelt, dass der Kurs aus gegebenen Gründen nicht anzeigbar ist. Das erhöht die Benutzbarkeit der Webanwendung signifikant, weil der Benutzer sofort einen Fehler seinerseits ausschließen kann. Die Fehleranzeige wird zusätzlich auch dafür benutzt *Appointments* anzuzeigen, welche nicht im Stundenplan anzeigbar sind, wie zum Beispiel Kurse zu unüblichen Uhrzeiten oder am Wochenende. Dies erfolgt mit schwarzer Schrift, da es sich nicht um einen Fehler handelt. Zusätzlich wird die Modulfarbe angezeigt um einen visuellen Bezug zu anderen Kursen herzustellen.

4.5 Entwicklung der Webanwendung

Dieser Abschnitt befasst sich mit der Implementierung der zuvor erstellten Funktionalitätenliste, dabei werden die auftretenden Probleme und die Gründe für die letztendliche Struktur in die Mitte der Betrachtung gerückt.

4.5.1 Erweiterung des Studienverlaufplaners

Als erster Schritt sind die Klassen *main.js* und *ModuleventComb.js* des Studienverlaufplaners dahingehend erweitert worden, dass falls ein Kurs an-

oder abgewählt wird, dieser mit Bezug auf das jeweilige Semester im SessionStorage abgespeichert wird. Hinzukommt in der Klasse *TableUtl.js* ein Button, der sich unter dem Semesternamen befindet und den Benutzer auf den Wochenverlaufplaner weiterleitet. Die benötigten Informationen werden über Parameter in der URL[13] weitergereicht. Dazu gehören die *SemesterID*, die *studyRegulationNumber*, welche die Studienordnung definiert, und der Name des ausgewählten Semesters. Um die von mir vorgenommenen Änderungen besser nachvollziehen zu können werden alle nicht von mir bearbeiteten Codestellen mit dem String „added by Martin Schlegel“ erkenntlich gemacht.

4.5.2 Laden der *Course*-Daten

Auf Seiten des Servers wird in der *startWeekSchedule(...)* Methode, welche den Einstieg in den Wochenverlaufplaner darstellt, die zugehörige *schedule.html* Datei zurück gegeben. Den Einsteigspunkt in die clientseitige Komponente stellt die Klasse *WeekScheduleMain.js* dar. In dieser werden zunächst mittels der übergebenen *SemesterID* die *CourseIDs* aus dem SessionStorage geladen. Diese werden dann innerhalb der Funktion *getEventsData(courseIds)* mittels einer asynchronen AJAX-Anfrage an den Server übergeben. Dieses löst im Server die Funktion *getEvents(...)* aus. In dieser Funktion werden alle benötigten Information aus der Datenbank des MVS zusammen getragen und mittels JSON[14] an den Clienten übertragen. Diese Daten werden mittels der Funktion *processReceivedData(data)* auf Seiten des Clients in den lokalen Datenstrukturen abgespeichert, siehe Abbildung 7 auf Seite 18. In der Endphase des Projektes wurde entschieden, dass die Anwendung veranstaltungs- und nicht modulorientiert sein soll. Dies wirkte sich mit einer Veränderung der Datenstruktur und der Ladefunktionalität aus. Die Änderung der Datenstruktur bewirkte eine komplizierte Umbildung der gesamten Anwendung, da jeder Zugriff auf die Datenstruktur angepasst werden musste. Dieser Fehler hätte sich leicht durch eine genauere Anforderungsermittlung zu Beginn des Projektes vermeiden lassen.

4.5.3 Visualisierung der Veranstaltungen

Die Anzeige der geladenen Daten stellt den nächsten großen Schritt dar. Der Stundenplan soll dabei sehr viele Methoden bereitstellen um alle geforderten Funktionalitäten im gewünschten Maß umzusetzen, wie zum Beispiel die Methoden *addToScheduleEventList(eventList, enabledEvents)* und *addToScheduleEvent(eventData)* um neue Events in den Stundenplan einzutragen. Die Methode *insertAlgorithmEvents(eventList)* de- und aktiviert die

[13] Uniform Resource Locator
[14] JavaScript Object Notation

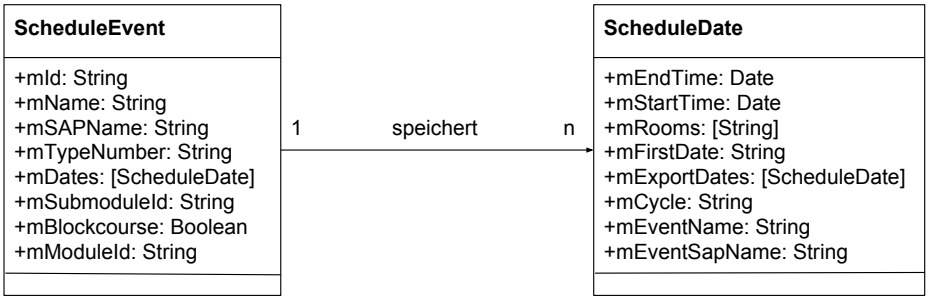

Abbildung 7: UML-Klassendiagramm der clientseitigen Datenstrukturen

durch den Algorithmus berechneten *Appointments*. Die Methode *removeAllEventsFromSchedule()* und *removeEventFromSchedule(event)* entfernen die *Events* wieder aus dem Stundenplan. In der ersten Implementierung war nur das Eintragen von zweistündigen Veranstaltungen möglich. Daraus resultierte auch die Designentscheidung, dass mehrere *Appointments* in einer Zelle untereinander angeordnet wurden, was viele Schritte in der restlichen Implementierung vereinfachte. Das erst nachträgliche Ändern auf einstündige Zeitslots lässt sich wiederum auf eine fehlerhafte Anforderungsermittlung zurückzuführen und bewirkte einen vollständigen Umbau der entstandenen Stundenplanstruktur um diesen auch für ein- oder mehrstündige Veranstaltungen verfügbar zu machen. Diese Änderung war nicht so tiefgreifend wie die nachträgliche Änderung der Datenbank, da sie nur den Stundenplan betraf. Als Resultat ist die im Designabschnitt Stundenplan beschriebene Struktur entstanden.

4.5.4 Manuelle Aktivierung der Termine

Die Zellen im Stundenplan wurden um zwei Methoden, *enableClickable(innerTd)* und *disableClickable(innerTd)*, erweitert, welche das Aktivieren und Deaktivieren ermöglichen. Ein Problem bei der Umsetzung war, dass alle zu einem *Appointment* gehörenden Zellen aktiviert oder deaktiviert werden mussten. Um dies umzusetzen wird durch die über- und unteren Zellen traversiert um zugehörige Zellen zu finden und gegebenenfalls deren Status zu verändern. Als aktivierte Zelle wird immer die oberste eines *Appointments* in *gActiveDates* anhand ihrer *submoduleID* abgespeichert.

4.5.5 Visualisierung zusammengehörender Termine

Um die verschiedenen Termine auch visuell in Abhängigkeit zueinander zu setzen, sollten diese eingefärbt werden. Zunächst sind mittels CSS verschiedene statische Farbklassen erstellt und benutzt worden. Der Tatsache geschuldet, dass die maximale Anzahl der benötigten Farben und die manuell

gesuchten Farben nicht den Anforderung entsprachen, wurde eine Lösung gebraucht, welche automatisch die benötigten Farben dynamisch generiert. Inspiration für eine Umsetzungsmöglichkeit ist auf [ranhr] zu finden. Die erzeugten Farben müssen die folgende drei Eigenschaften erfüllen.

1. Auf den Farben muss schwarze Schrift gut lesbar sein.
2. Die Farben sollen sich möglichst stark untereinander unterscheiden.
3. Die Anzahl ist initial nicht klar definiert.

Die erste Eigenschaft lässt sich mit der Verwendung des HSV[15] Farbraums bewerkstelligen. Dabei werden die Sättigung (**S**aturation) und der Hellwert (**V**alue) statisch gesetzt und die resultierenden Farben nur durch das modifizieren des Farbwerts (**H**ue) verändert. Die verbleibenden beiden Anforderungen lassen sich durch die Verwendung des goldenen Schnitts als Schrittweite für den Hue umsetzen. Diese gesamte Funktionalität wurde in der Klasse *ColorGenerator.js* zusammengefasst.

4.5.6 Anzeige von nicht eintragbaren Veranstaltungen und Terminen

Unter Umständen kommt es dazu, dass manche anzuzeigende Daten fehlerhaft sind. Um dem Benutzer diese Fehler anzuzeigen gibt es die Methode *addToErrorLegend(colorClass, eventName)*. Um *Appointments* anzuzeigen, welche am Wochenende stattfinden oder zu ungewöhnlichen Uhrzeiten abgehalten werden gibt es die Methode *addToSpecialDateLegend(...)*. Um angezeigte Daten wieder auszublenden werden die *removeFromErrorLegend(eventName)* und *removeAllFromErrorLegend()* bereit gestellt.

4.5.7 Entwicklung des Schedulingalgorithmus

Die gesamte Logik des Schedulingalgorithm befindet sich in der Klasse *ScheduleAlgorithm.js*. Dieser Klasse werden alle momentan im Stundenplan eingetragenen *Course* übergeben. Diese *Course* werden zunächst nach *staticDates* und *maybeDynamicDates* sortiert, wobei in diesem Schritt nur Vorlesungen als *staticDates* angesehen werden. Anschließend werden die *maybeDynamicDates* wiederum in *staticDates* und *dynamicDates* aufgeteilt. Existiert zu einer *SubmoduleID* nur ein *maybeDynamicDate* wird dieses den *staticDates* zugeordnet, anderenfalls wird es unter den *dynamicDates* abgespeichert. Nun werden alle *staticDates* in den virtuellen Stundenplan des Algorithmus eingetragen.
Im nächsten Schritt werden alle möglichen Permutationen der *dynamicDates* nacheinander ausprobiert und jeweils mittels einer Gewichtsfunktion bewertet. Dabei wird der Stundenplan mit dem höchsten Gewicht gespeichert

[15] Hue-Saturation-Value

und letztendlich an den realen Stundenplan zurückgegeben. Mittels der Gewichtsfunktion lassen sich die Ergebnisse des Algorithmus beeinflussen. Das Gewicht eines Stundenplans setzt sich folgendermaßen zusammen. Zunächst werden alle Einzelgewichte der Zellen addiert, dazu kommt eine negative Bewertung für parallel stattfindende *Appointments*. Finden an einem Tag keine Veranstaltungen statt fließt dies positiv in die Bewertung ein, wobei Montag und Freitag positiver verrechnet werden. Befinden sich an einem Tag mehr als sechs Stunden *Appointments* wird dies negativ Bewertet. Als letzter Punkt gibt es einen Abzug für freie Zellen zwischen belegten Zellen. Die Einzelgewichte der Zellen setzen sich aus den übergeben dynamischen Gewichten des Benutzers und den Standartgewichten für die Zeitslots zusammen. Die Standartgewichte sind dabei an der physiologischen Leistungsbereitschaft des durchschnittlichen Menschen angepasst, siehe Abbildung 9 auf Seite 28 und siehe Abbildung 10 auf Seite 29. Die Standartgewichte, welche der Algorithmus in seinen Berechnungen verwendet sind in der Abbildung 11 auf Seite 30 abgebildet.
Im Folgenden kommt ein kleines Beispiel, welches die Gewichtsberechnung verdeutlicht. In Abbildung 8 auf Seite 21 ist eine mögliche Kurs- und Gewichtsbelegung der dynamischen Gewichte zu sehen auf die sich diese Beispielberechnung bezieht.

1. Die dynamischen- und Standartgewichte der belegten Zeitslots werden aufsummiert (Montag, Dienstag, Dienstag und Donnerstag).
   $25 + (-1985) + 50 + 2025 = 65$
2. Die negative Bewertung paralleler *Appointments* (Dienstag).
   -1000
3. Die positive Bewertung freier Tage (Mittwoch und Freitag).
   $500 + (500 + 100) = 1100$
4. Die negative Bewertung von Tagen mit mehr als sechs belegten Zeitslots (nicht vorhanden).
5. Die negative Bewertung von Freistunden (Dienstag).
   -100
6. Die Summe aller Einzelergebnisse:
   $65 + (-1000) + 1100 + -(100) = 65$

4.5.8 Aus- und Einblenden von nicht aktivierten Terminen

Das Aus- und Einblenden wird in den beiden Methoden *hideDisabledDates()* und *showDisabledDates()* durchgeführt. Dabei wird über jede Zelle beziehungsweise *Appointment* des Stundenplans traversiert und diese ausgeblendet beziehungsweise eingeblendet, falls sie einen beziehungsweise keinen Alphawert besitzten.

Zeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
08:00 - 09:00	SG:15 DG: 0	SG:15 DG: -2000 **Ü1, Ü2**	SG:15 DG: 0	SG:15 DG: 0	SG:15 DG: -2000
09:00 - 10:00	SG:25 DG: 0 **VL1**	SG:25 DG: 0	SG:25 DG: 2000	SG:25 DG: 2000 **VL3**	SG:25 DG: 0
10:00 - 11:00	SG:50 DG: 0	SG:50 DG: 0 **VL2**	SG:50 DG: 0	SG:50 DG: 0	SG:50 DG: 0

Abbildung 8: Beispielbelegung eines Stundenplanausschnittes, SG: statisches Gewicht, DG: dynamisches Gewicht

4.5.9 Entfernung einzelner Veranstaltungen

In der Methode *addToEventSelectionModalBody(...)* werden alle *Course* in der *eventList* und in den *modalBody* eingetragen. Zusätzlich wird für jede angelegte Checkbox eine *onChange()* Funktion angelegt. In dieser wird beim Anklicken eines Events der aktuelle Status getauscht und mittels der Methode *updateSchedule(index, semesterEvents)* an den Stundenplan weitergegeben. In dieser werden wiederum die *Course* aus dem Stundenplan entfernt oder eingefügt. Da der Algorithmus beim Ausführen auf die aktuell eingetragenen *Appointments* beziehungsweise *Course* zugreift reicht dies schon aus um einzelne Veranstaltungen nachträglich zu entfernen. Bei dem Ausblenden des Modals werden wieder alle Eintragungen gelöscht.

4.5.10 Dynamisierung des Schedulingalgorithmus

Wird das Modal zur Dynamisierung das erste mal aufgerufen wird die Funktion *initModalSchedule()* ausgelöst, welches den Status jeder Zelle auf *maybe* setzt. Mittels der Funktionen *activateYes(newActiveButton)*, *activateMaybe(newActiveButton)* und *activateNo(newActiveButton)* kann der Status einzelner Button, ganzer Tage und Zeitslots verändert werden. Wenn das Modul geschlossen wird werden in der Funktion *evaluateAdaptedWeights()* die angepassten Gewichte aus den jeweiligen Buttonstatus ermittelt. Diese werden dann wiederum beim Starten des Algorithmus mittels der Variablen *gDynamicWeights* diesem übergeben.

4.5.11 Export des Stundenplans

Um den Stundenplan als ICS[16] Datei herunterzuladen wurde ein externes Framework Namens *ics.js* verwendet, dieses verwendet wiederum das *Blob.js* und *FileSaver.js* Framework. Durch das Verwenden dieser Klassen muss in der Exportfunktion nur über den Stundenplan traversiert werden um alle aktivierten *Appointments* dem Framework mit der Methode *addEvent()* hinzuzufügen. Anschließend werden in der Methode *getSpecialDates()* alle

[16]Internet Calendaring and Scheduling Core Object Specification

nicht eintragbaren und dennoch aktiven *Course* ausgelesen und ebenfalls dem Framework hinzugefügt. Im letzten Schritt wird der Stundenplan als ICS Datei mittels der Funktion *download()* heruntergeladen. Dabei mussten für den Safari Browser im benutzen Framework kleine Anpassungen getätigt werden um es auch dafür kompatibel zu gestalten.

5 Schlussfolgerung

Dieses Kapitel fasst das Projekt zusammen und gibt einen Ausblick, welche Ergänzungsmöglichkeiten existieren um den Wochenverlaufplaner weiter zu verbessern und zu erweitern.

5.1 Fazit

Im Großen und Ganzen sind alle gesetzten Ziele erfüllt worden. Ob das Design perfekt auf die Anforderungen abgestimmt ist ist nicht evaluiert, da leider innerhalb des Projektes nicht genügend Zeit zur Verfügung stand um einen Benutzbarkeitstest mit Studenten durchzuführen. Der Wochenverlaufplaner übernimmt die ausgewählten Veranstaltungen vom Studienverlaufplaner und zeigt diese veranstaltungsorientiert in einem Stundenplan an. Dabei kann der Benutzer manuell den Stundenplan bearbeiten und einen Algorithmus benutzen. Dieser ist durch den Benutzer zusätzlich auf dessen Bedürfnisse anpassbar, indem der Benutzer mittels einem Modal die Zeitslots in drei Kategorien einordnen kann. Am Ende ist es möglich den Stundenplan als ICS Datei herunterzuladen und somit in andere Kalendersoftware zu importieren. Dabei sind alle Funktionen ohne Anmeldung benutzbar.
Die Durchführung des Projektes bis zu dem jetzigen Entwicklungsstand war anspruchsvoll und somit auch sehr lehrreich. Der schwierigste Abschnitt des Projektes stellte die Umsetzung der Funktionalitäten in JavaScript dar, da am Anfang der Arbeit keinerlei Erfahrungen in diesem Bereich zur Verfügung standen. Dadurch und Aufgrund des gewählten iterativen Entwicklungsprozesses, der mehrmals zu nachträgliche Veränderungen führte, ist die clientseitige Architektur mit Sicherheit verbesserbar. Allein schon die Tatsache, dass die Klasse *WeekScheduleMain.js* mehr als 1000 Zeilen Code besitzt, bringt diesen Umstand zum Ausdruck. Es sollte ein Refactoring der JavaScript Klassen durchgeführt werden, vor allem mit dem Ziel den Code in mehrere Klassen aufzuteilen und somit übersichtlicher zu gestalten. Mitunter kommt es dazu, dass der Server beim Start der Anwendung merkliche Antwortzeiten hat. Dies ließe sich durch spezialisiertere SQL[17]-Anfragen an die Datenbank verbessern. Der implementierte Schedulingalgorithmus stellt einen klassischen Brute-Force Algorithmus dar, welcher zusätzlich optimiert werden kann um geringere Rechenzeiten zu erzielen.

5.2 Ausblick

Im folgenden Abschnitt werden einige Funktionalitäten vorgestellt, welche den Wochenverlaufplaner noch attraktiver für die Zielgruppe gestalten können.

[17] Structured Query Language

- Zu aller erst sollte ein Benutzbarkeitstest mit Studenten aus dem ersten Semester durchgeführt werden um zu testen, ob das Design auf Zustimmung trifft, die Anwendung die Erwartungen erfüllt und um zusätzliche Erweiterungsideen zu erhalten.
- Das Anbieten von weiteren Informationen zu den Kursen ähnlich wie im Studienverlaufplaner würde den Wochenverlaufplaner gewinnbringend erweitern.
- Für viele Studenten ist der leitende Tutor zu einem Übungstermin wichtiger als der konkrete Termin, aus diesem Grund wäre es hilfreich bei Übungsterminen den Tutor mit anzuzeigen. Dies ist momentan nicht möglich, da die Datenbank diese Informationen nicht bereitstellt.
- Momentan werden Termine außerhalb von 8 bis 20 Uhr und am Wochenende als Text seperiert über dem Stundenplan angezeigt. Besser wäre es den Stundenplan dynamisch um die benötigten Zeiträume zu erweitern. Somit würden leere Zeitslots wegfallen und die gesamte Anwendung übersichtlicher erscheinen.
- Wenn sich innerhalb einer Zelle mehrere aktivierte *Appointments* befinden sollte der Benutzer darauf visuell hingewiesen werden.
- Da die zugrunde liegende Datenbank des MVS auf 30 Minuten Slots angepasst ist, sollten diese auch im Stundenplan darstellbar sein.
- Der Wochenverlaufplaner könnte auf andere Informationen des KVV[18] zugreifen, wie zum Beispiel sollten ausgebuchte Übungstermine im Algorithmus schlechter bewertet werden um automatisch eine gleichmäßige Auslastung der angebotenen Termine zu fördern.
- Wenn es dem Benutzer möglich wäre sich anzumelden, wäre es denkbar seine ausgewählten Kurse und Übungstermine in sein zugehöriges KVV zu importieren und somit dem Benutzer doppelte Arbeit zu ersparen.

5.3 Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei meiner Bachelorarbeit und während meines Studiums unterstützt haben. Mein Dank gilt meinen Eltern Ines Haferkorn und Karsten Schlegel für die Unterstützung während meines Studiums. Ein besonderer Dank geht an meine Betreuerin Prof. Dr. Margarita Esponda-Argüero und Stephan Sundermann für die kontinuierliche Unterstützung und bei auftretenden Problemen und Fragen zum MVS.

[18] Kommentiertes Vorlesungsverzeichnis

Referenzen

[chrhr] `http://wiki.iao.fraunhofer.de/index.php/Chronobiologische_Arbeitsgestaltung#Rhythmisches_System_des_Menschen`, 29.01.2016 17.33 Uhr.

[Joh79] Michael R. Garey David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W H Freeman Co, 1979.

[JZ00] B. Knab J. Zulley. *Unsere innere Uhr.* Herder Freiburg, 2000.

[kaphr] `http://ls11-www.cs.uni-dortmund.de/lehre/AE/kap_lopopt.pdf`, 16.02.2016 11.18 Uhr.

[KM00] Detlef Gröger Kurt Marti. *Optimierungsprobleme.* Physica-Verlag HD, 2000.

[ranhr] `www.martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/`, 04.01.2016 18.54 Uhr.

[sprhr] `www.docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html`, 08.12.2015 20.55 Uhr.

[thyhr] `www.thymeleaf.org/documentation.html`, 19.12.2015 16.47 Uhr.

[w3shr] `www.w3schools.com`, 21.01.2016 14.39 Uhr.

[wirhr] `http://wirtschaftslexikon.gabler.de/Definition/greedy-algorithmus.html`, 01.04.2016 17.00 Uhr.

Abbildungsverzeichnis

Abkürzungsverzeichnis

MVS Modulverwaltungssystem

HTML Hypertext Markup Language

CSS Cascading Style Sheets

PHP Hypertext Preprocessor

HTTP Hypertext Transfer Protocol

URL Uniform Resource Locator

AJAX Asynchronous JavaScript and XML

JSON JavaScript Object Notation

KVV Kommentiertes Vorlesungsverzeichnis

ICS Internet Calendaring and Scheduling Core Object Specification

ER Entity-Relationship

MVC Modell-View-Controller

SQL Structured Query Language

HSV Hue-Saturation-Value

6 Anhang

6.1 Abbildungen

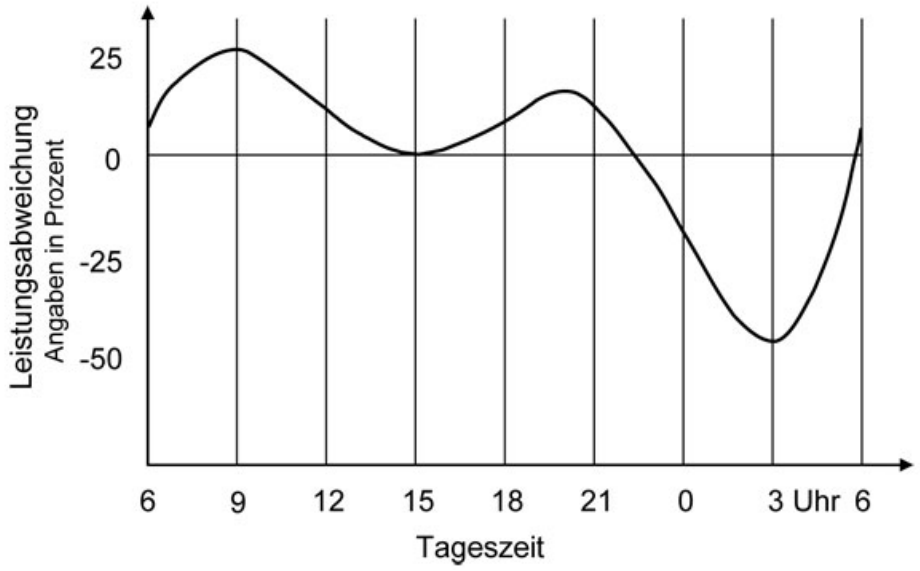

Abbildung 9: Schema der physiologischen Leistungsbereitschaft im Tagesverlauf (nach Hildebrandt et al. 1998) Quelle: [chrhr]

Tageszeit	*Empfohlene Tätigkeit*
7-8 Uhr	Der Körper liefert Energie für die Tagesarbeit; Weckzeit
10-11 Uhr	Kreativität, Konzentration und Kurzzeitgedächtnis sind optimal
11-12 Uhr	Energiehöhepunkt, Sehen und Rechnen sind optimal
12-13 Uhr	Die Leistungsfähigkeit sinkt; Zeit für das Mittagessen
13-14 Uhr	Tagestief; erhöhte Schlafbereitschaft des Körpers
14-15 Uhr	Ideale Zeit für die Siesta
15-16 Uhr	Tageshöhepunkt; das Langzeitgedächtnis ist wach
17-18 Uhr	Ideale Zeit für Sport; Organismus ist gut durchblutet
18-19 Uhr	Tagesrückblick; Entspannung für die Nacht

Abbildung 10: Ablauf eines Arbeitstages aus chronobiologischer Sicht (nach Zulley/ Knab 2000) Quelle: [chrhr]

Art	Gewicht
Standartgewichte (Zeitslots)	
08:00 - 09:00 Uhr	15
09:00 - 10:00 Uhr	25
10:00 - 11:00 Uhr	50
11:00 - 12:00 Uhr	50
12:00 - 13:00 Uhr	25
13:00 - 14:00 Uhr	0
14:00 - 15:00 Uhr	0
15:00 - 16:00 Uhr	40
16:00 - 17:00 Uhr	40
17:00 - 18:00 Uhr	25
18:00 - 19:00 Uhr	0
19:00 - 20:00 Uhr	0
parallele Appointments	-1000
freier Tag	500
freier Montag/ Freitag- Bonus	100
mehr als 6 Stunden Appointments pro Tag	-300
freie Zellen (Freistunde)	-100
dynamische Gewichte	
ja	2000
(ja)	0
nein	-2000

Abbildung 11: Standartgewichte des Algorithmus