Erstellen einer Vertretungsplan-Webseite

Kamal Abdellatif
Tim Gerlach

17. Mai 2016

Fachbetreuer:
Herr König

Inhaltsverzeichnis

1 Einleitung

Der Vertretungsplan hat schon seit längerer Zeit eine Online-Präsenz, welche intensiv von den Schülern dieses Gymnasiums genutzt wird. Doch nach Nachfragen unter den Schülern erkannten wir, dass eine Überarbeitung des Vertretungsplans notwendig geworden war. Vor allem der hochfrequenten Nutzung durch Mobilgeräte ist die aktuelle Version der Webseite nicht gewachsen. Daher nahmen wir uns im Rahmen des Informatikprojektes der zehnten Klasse eine Neugestaltung vor. Dabei setzen wir unsere Schwerpunkte auf Optimierung für mobile Endgeräte, Benutzerfreundlichkeit für sowohl Schüler als auch Lehrer und innovatives, responsives Design. Der Vertretungsplan sollte lesbarer, intuitiver und logischer strukturiert sein, damit man als Schüler alle wichtigen Informationen auf einen Blick erkennt.

2 Theoretische Grundlagen

Wir verwenden für den Aufbau einer funktonierenden Kommunikation viele verschiedene Arten von Code, Dateiformaten und Übertragungsprotokollen. Ein kleiner Teil des Datenaustauschs ist zur Veranschaulichung hier erklärt.

Zunächst liegen die Daten als XML-Code, also durch sogenannte Tags gegliederter Text vor. Ein wichtiges Prinzip beim Generieren der Website ist das Umwandeln dieser in ein für den Server benutzbares Format. Wir haben uns dazu entschieden, für diesen Vorgang `Regular Expressions`, kurz `regexp` zu benutzen: Ein Platzhalterstring wird auf eine Quelle „gematcht“, d.h. alle Strings, die auf ein entsprechendes Muster passen, werden mit einer Schlüsselbezeichnung zurückgegeben. So wird der reguläre Ausdruck `/^[\t]*\$/` auf alle nur mit Leerzeichen und Tabs gefüllten Zeilen passen. Damit lassen sich verschachtelte Klassenangaben wie bsp. `10A/ 10ABCET1` auftrennen: die Ethikgruppe 1 der Klassen 10a, 10b und 10c ist hier gemeint.

Der XML-Code wird so Schritt für Schritt in eine JSON-Datei (`JavaScript Object Notation`) umgewandelt. Dieses ist ein weit verbreitetes Speicherformat, welches Daten in Schlüsseln und Werten dartellt, die durch Listen und Gruppen strukturiert sind.

Die ursprüngliche Quelle für das XML-Dokument liegt hier bei einem Stundenplanverwaltungsprogramm der Schule. Die dort generierten Dateien werden über die `/upload` Webseite hochgeladen. Dafür werden jQuery und AJaX verwendet. jQuery ist eine beliebte JavaScript Bibliothek, die dem Anwender viel Arbeit wie das manuelle Registrieren von EventListenern, die unter anderem für das Abfangen von Mausbewegungen oder Tastendrücken sorgen können, erspart. AJaX steht für „Asynchronous JavaScript and XML“und bedeutet, dass über JavaScript die Seite mit dem Server kommunizieren kann, ohne eine komplette HTML-Seite als Rahmen zu übermitteln. Damit können sämtliche Dateien problemlos auf den Server übertragen werden.

Nach dem Hochladen und dem Konvertieren sind die Dateien bereit, vom Client abgerufen zu werden. Dafür müssen sie in einen von einem Browser interpretierbaren Code umgewandelt werden, was in fast allen Fällen HTML, also `Hypertext Markup Language` ist. Dafür benutzen wir den Template-Parser von Pyramid, einem Serverframework. Es werden in einem HTML-Dokument Angaben gemacht, die vom Server mit entsprechenden Informationen ersetzt werden. Dabei wird das zuvor generierte JSON-Dokument verwendet.

Zusammen mit Stildateien und Scriptcode werden die Informationen an den Client übertragen. Dafür muss dieser zunächst ein Passwort angeben. Noch auf der Clientseite wird daraus eine Prüfsumme erstellt, aus der das ursprüngliche Passwort mit aktuellen technischen Möglichkeiten nicht ableitbar ist. Diese wird auch als `Hashsum` bezeichnet, was so viel wie „zerhackter Wert “bedeutet. Ein solcher wird mithilfe eines Requests an den Server übertragen, der die Prüfsumme mit einer Liste vergleicht und eine positive oder negative Rückmeldung an den Client sendet.

Vor dem Auslesen des Passworts ist der Kommunikationsweg zwischen Server und Client sicher, doch ein weiteres Risiko bleibt bestehen: Wer den Hashwert abfängt, kann sich als Client ausgeben und diesen direkt übertragen. Die einzige Lösung dagegen ist, den kompletten Datenaustausch End-to-End zu verschlüsseln. Eine Lösung hierfür ist SSL, welches durch ein etwas anderes Protokoll, `https` statt `http`, genau dies gewährleistet.

3 Lösungsidee

3.1 Wahl eines Webframeworks

Für die Umsetzung des Projektes war ein Webframework unabdingbar, welches einem Anforderungen wie Sicherheit, ein voll funktionsfähiger Webserver, Template-Rendering, Cookie-Policies und vielem mehr abnimmt. Jedoch waren viele Frameworks viel zu umfangreich, und erforderten viel überflüssigen Aufwand. Daher bat sich ein Microframework an. Auf Grund von unseren Vorkenntnissen und der schnellen Produktion entschieden wir uns für ein `Python`-basiertes Microframework. Schlussendlich fiel die Wahl auf das `Pyramid Web Framework`[1].

3.2 Struktur der Applikation

Die Applikation ist in drei Teile gegliedert: die `Python`-Scripts, den `Pyramid Web Server` und die eigentliche Seite. Sowohl die Scripts als auch der Webserver sind serverseitig, während die Seite beim Client agiert.
Die Seite muss eine Nutzeroberfläche anbieten, die die Daten vom Server dem Nutzer visualisiert, und die vom Nutzer eingegebenen Daten dem Server zustellt. Der `Pyramid Web Server` muss für sicheren den Datenaustausch zwischen den `Python`-Scripts und der Seite via sorgen. An höhster stelle stehen die `Python`-Scripts, welche die vom Nutzer eingegebenen Daten bearbeiten, speichern, und allen Nutzern zurückgeben müssen.

3.3 Server

3.3.1 Funktionen

Der Server der Webapplikation muss mehrere Aufgaben bewältigen. Ein Webserver muss `HTTP` bedienen können, um dem Browser des Nutzers eine `HTML`-Oberfläche bieten zu können. Dazu müssen Vetretungsplan-Daten auf dem Server gepeichert, abgerufen, und bearbeitet werden. Weiterhin muss der Server in der Lage sein, vom Nutzer hochgeladene Dateien zu empfangen, konvertieren und zu seinen gepeicherten Daten hinzuzufügen. Damit nur autorisierte Nutzer Daten bearbeiten können, ist eine sichere Authenifizierung über eine Passworteingabe des Nutzers erforderlich. Diese erforderliche Web-Sicherheit beinhält Hashwerte und die Verwendung von `SSL`.

[1] siehe `http://docs.pylonsproject.org/projects/pyramid/en/latest/index.html`

3.3.2 Struktur

Der Server baut auf den schon von Pyramid gegebenen Strukturen auf. In dem Webframework sind schon viele Teile eines Servers gekapselt, sodass wir uns nicht direkt damit außeinandersetzen mussten. Dies umfasst Authentifikation, also das Erkennen und Speichern von Nutzern, Autorisierung, die Einschränkung aller Nutzer auf ihre Erlaubnisbereiche. Weiterhin nimmt uns das Framework das gesamte Kommunikationsprotokolle `HTTP` ab. Somit läuft die gesamte Kommunikation mit dem Client über Pyramid ab. Dies umfasst das Senden von Daten via `GET`-Requests, das Empfangen von Daten durch `POST`-Requests und die Kommunikation via `AJaX`.
Am Back End des Servers befinden sich die Python-Scripts. Diese haben Zugriff auf die auf dem Server gespeicherten Daten, und steuern das Framework. Die gesamte applikationsspezifische Programmlogik wird innerhalb dieser Scripts ausgeführt. Die Scripts bekommen Nutzerdaten von Pyramid in Form Requests (Anfragen) vom Nutzer. Diese verarbeiten diese je nach Art des Requests, bearbeiten gegebenenfalls die gespeicherten Daten auf dem Server, und geben dem Framework alle Informationen für eine Antwort an den Client. Meist bestehen diese Informationen aus einzelnen Variablen.
Um dem Nutzer jedoch eine Antwort zu senden, müssen diese Daten noch in ein dem Nutzer verständlichen Format wie `HTML` gebracht werden. Diese Aufgabe wird von sogenannten Renderern erledigt. Renderer setzen aus einer Template (Vorlage), und den speziefischen Daten eine vollständige Antwort zusammen. Für jeden Datensatz von den Python-Scripts, welcher an den Client gesendet werden soll, gibt es eine bestimmte Template. So können zum Beispiel mit Leichtigkeit große, komplexe HTML-Seiten generieren, indem man unveränderliche Bestandteile in Templates auslagert, und nur die veränderlichen Daten der Scripts vom Renderer einsetzen zu lassen. Handelt es sich nur um unveränderlichen Daten in einer Template, wenn es sich beispielsweise um ein reines `CSS`-Stylesheet handelt, so ist kein Renderer von nöten. Dies wird durch sogenannte Static Assets realisiert, welche ganze Dateien unbearbeitet dem Nutzer zur Verfügung stellen.
Die Applikation an sich läuft auf einem Unix-Server mit `nginx`, welches die Basis für die `HTTP`-Kommunikation für Pyramid bildet.

3.3.3 Web-Sicherheit

Die Sicherheit des Systems steht an erster Stelle. Daher haben wir mehrere Maßnahmen ergriffen, um unsere Applikation so sicher wie möglich zu gestalten. Die Kommunikation zwischen Server und Client geschieht eigentlich via `HTTPS`. Dabei handelt es sich um die schon erwähnte `HTTP`-Kommunikation, gekoppelt mit `SSL`, ein `RSA`-verschlüsseltes Protokoll, welches ein server-spezifisches Sicherheits-Zertifikat benötigt, und globaler Standart für Verschlüsslung im Internet ist. Dieses Protokoll ist in `nginx` implementiert, und versichert dem Nutzer, dass kein Dritter Zugriff empfindliche Daten hat.

Die Passwörter für die verschiedenen Bereiche dürfen nur den jeweils Berechtigten bekannt sein. Dies schließt ein Speichern der Passwörter auf dem Server in Klartext aus,

da selbst wir als Programmierer uns nicht erlauben dürfen, den Vertretungsplan zu bearbeiten. Weiterhin dürfen die Passwörter trotz `SSL`-Verschlüsslung auch nicht in Klartext gesendet werden, da dies impliziert, dass sie im Klartext als Wert irgenteiner Variable auf den Server-Scripten auszulesen sind. Daher werden die Passwörter schon auf der Client-Seite von JavaScript in sogenannte Hashwerte umgewandelt.

3.3.4 Bearbeitungsprozess

Um den Vertretungsplan zu aktualisieren, werden vom verantwortlichen Lehrer ein oder mehrere XML-Dateien auf den Server hochgeladen. Dazu ist eine Anmeldung mit dem Bearbeitungs-Passwort erforderlich. Der Server liest daraufhin alle Dateien aus, wobei jede Datei die Informationen für einen Tag beinhält, und fügt die Daten dem Gespeicherten hinzu oder ersetzt diese. Dabei müssen die Daten in XML-Form in unsere Datenstruktur übertragen werden, und im JSON-Format gespeichert werden. Dieser Prozess ist der komplexeste in der Applikation und besitzt einen strikten Ablauf.
Dieser ist in folgende Schritte gegliedert:

1. Abrufen XML-Daten
2. Auslesen XML-Daten
3. Konvertieren
4. Auslesen der gespeicherten Daten
5. Hinzufügen
6. Speichern der Daten
7. Antwort formulieren

Das Verarbeiten einer hochgeladenen Datei fängt mit dem Auslesen der Datei an. Die Daten werden in eine temporäre Datei binär überschrieben, und mit entsprechendem UTF-8 Encoding ausgelesen. Somit sind die Daten zur weiteren Bearbeitung nur in Form eines String-Buffers in der Laufzeitumgebung von Python vorhanden. Zeigt die hochgeladene Datei ins Leere, indem zum Beispiel der POST keine Datei beinhält, wird ein FileNotFoundError geworfen. Konnten die Daten nicht in UTF-8 dekodiert werden, kommt ein UnicodeDecodeError. Beide Fehler werden als Fehlerstufe `ERROR` eingestuft (*siehe Fehlerbetrachtungen*).
Darauf hin werden die in einem String gespeichterten XML-Daten geparst, und in ein ElementTree -Objekt umgewandelt. Dies übernimmt das `xml.etree` Modul der Python-Standardbibliothek. Auch hier kann ein als `ERROR` eingestufter Fehler auftreten, falls die XML-Daten z. B. auf Grund fehlerhafter Syntax nicht geparst werden konnten.
Die Daten für einen Tag werden daraufhin aus dem ElementTree-Objekt über den komplexen Konvertierungsprozess in ein Python-Dictionary umgewandelt, welches fast identisch mit dem Datenspeicherungsformat JSON ist. Diese Konvertierung baut auf der einheitlichen Struktur der XML-Daten auf, und kann somit bei semantischen Fehlern in den Daten einen Fehler der Stufe ERROR hervorrufen.
Dieser Datensatz für einen Tag soll nun dem gespeicherten Daten hinzugefügt werden.

Dazu müssen erst die alten Daten aus der gespeicherten Datei ausgelesen werden. Dabei können wieder ein FileNotFoundError , ein syntaktischer Fehler oder ein semantischer Fehler auftreten, welche alle `CRITICAL`-Fehler sind.
In diesen Datensatz, der wieder als Dictionary vorhanden ist, werden nun die neuen Daten hinzugefügt, ersetzt oder ggf. gelöscht. Dieser aktualisierte Datensatz wird daraufhin wieder in der JSON-Datei gespeichert und überschreibt die alten Daten in der Datei.
Mögliche `CRITICAL`-Fehler werden nun automatisch behandelt und dem Systemadministrator weitergegeben. Schlussendlich muss eine Antwort für den Client formuliert werden, welche ihn über das erfolgreiche bzw. misslungene Bearbeiten der Daten informiert.

3.3.5 Gegebene Datenstruktur

Eines unserer Ziele ist, den Vertretungsplan übersichtlicher und intuitiver zu gestalten. Dazu müssen die Informationen von Grund auf restrukturiert werden. Dazu setzten wir uns mit der Struktur der vorhandenen Daten auseinander und entwickelten einen Konvertierungsprozess, der eine für uns sinnvolle Datenstruktur erzeugt.

Struktur der XML-Daten

Die Daten können von dem Stundenplan-Verwaltungs-Programm unserer Schule in zwei verschiedenen Formen ausgegeben werden: den Schülerplan und den Lehrerplan. Aktuell wird der Schülerplan verwendet, jedoch entschieden wir uns für den Lehrerplan, da dieser mehr Informationen enthält und simpler strukturiert ist. Die Daten können uns in vielen verschiedenen Formaten ausgegeben werden. Wir entschieden uns für XML, da es gut mit Python auszulesen ist und einfach strukturiert ist. Jeder Tag wird nach einem ganz bestimmten Schema in einer XML-Datei repräsentiert. Im Dateikopf stehen allgemeine Informationen, von denen wir nur das Datum des betreffenden Tages auslesen. Darauf folgen weniger relevante Daten, welche eine Auflistung aller betroffenen Lehrer und Klassen sowie den Hofaufsichtsänderungen beinhalten. Diese ignorieren wir, da wir die enthaltenen Informationen auch aus den restlichen Daten herleiten können, und unser Vertretungsplan primär auf Schüler ausgerichtet ist. Der Hauptteil der XML-Datei ist eine Auflistung aller Änderungen (Aktionen) des Tages. Jede Aktion ist folgendermaßen strukturiert:

```
<aktion>
    <stunde></stunde> <!-- betreffende Stunde (Zeit) -->
    <fach></fach> <!-- normales Unterrichtsfach -->
    <lehrer></lehrer> <!-- normaler Lehrer -->
    <klasse></klasse> <!-- betroffene Klasse -->
    <vfach></vfach> <!-- Vertretungsfach -->
    <vlehrer></vlehrer> <!-- Vertretungslehrer -->
    <vraum></vraum> <!-- Vertretungsraum -->
    <info></info> <!-- Information als Text -->
</aktion>
```

Ein Beispiel wäre folgendes:

```
<aktion>
    <stunde>2</stunde>
    <fach>De</fach>
    <lehrer>MI</lehrer>
    <klasse>10C</klasse>
    <vfach>Ma</vfach>
    <vlehrer>MEL</vlehrer>
    <vraum>D209</vraum>
    <info>Aufgaben im, Unterrichtsraum</info>
</aktion>
```

Diese Aktion besagt, dass in der 2. Stunde das Fach Deutsch der Klasse 10c mit Frau Mittelstädt ersetzt wird durch Mathematik mit Herrn Mehlhos im Raum D209. Hat das Vertretungsfach, der Vertretungslehrer oder -raum den Wert „`---`“ oder gar keinen Wert, dann fällt das betroffene Fach aus.

Klassenbezeichner

Besonders Klassenbezeichner sind im Vertretungsplan schwer zu dechiffrieren. Daher nahmen wir die Syntax genau auseinander und bauten von Grund auf neue Klassenbezeichnner. Folgende Zeilen sind mögliche Bezeichner des Lehrer-Vertretungsplans:

```
10C
06A
08A,08B,08C/ 08FRZ2
06A,06B/ 06ABET
10C/ 10CIF2
11/ ma2
12/ ene
```

Wir unterscheiden zwischen drei Typen von Bezeichnern. Der erste Typ „`SIMPLE`“ ist in den ersten zwei Beispielen zu sehen, und steht für eine normale Klasse unter Stufe 11. Dabei gibt es zwei Teile: Klassenstufe (`grade`) und Unterstufe `subgrade`.

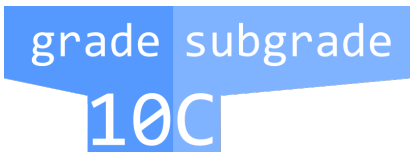

Abbildung 3.1: Aufbau eines einfachen Bezeichners

Den zweiten Typ nennen wir „`MULT`“ für „multiple“. Er bezeichnet alle Schülergruppen, welche nicht im Klassenverband an einer Stunde teilnehmen. Dies kann entweder

Klassen-intern wie das geteilte Fach wie Informatik sein, oder Klassen-übergreifend wie Fremdprachenfächer einer Klassenstufe. Er beginnt mit einer Aufzählung aller betroffenen Klassen (`targets`) in der Form von `SIMPLE`, und endet mit einer allgemeinen Bezeichnung. Diese fängt immer mit der Klassenstufe (`grade`) an, kann danach gefolgt sein von den betrefflichen Unterstufen (`subgrades`), hat danach immer das entsprechende Fach (`subject`) und endet ggf. mit einer Gruppennummer (`subclass`).

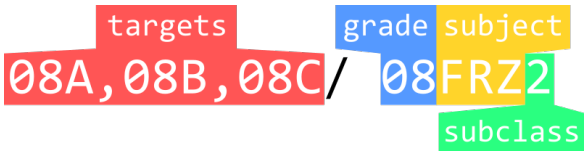

Abbildung 3.2: Aufbau eines Mehrfach-Bezeichners

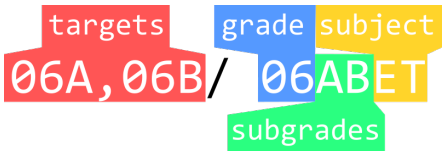

Abbildung 3.3: Aufbau eines Kurs-Bezeichners

Der dritte Typ „`COURSE`“ beschreibt alle Kurse. Er besteht immer Klassenstufe 11 oder 12, einem Kurs und einem Fach und einer optionalen Kursnummer am Ende.
Diese Unterteilung betrachtet jedoch keine AG's, WoU's, oder E- und Z-Kurse. Daher sind die Fächer an sich, wie sie als Element in jedem Bezeichner auftreten können, nochmals unterteilt. Ein Fach besteht dabei immer aus dem Namen als Abkürzung und einem optionalen Prefix und Suffix. So sind AG und WoU als Prefix und E- und Z-Kurse als Suffix implementiert. Dies ist allerdings noch erweiterbar.

3.3.6 Speicherformat

Um die ausgelesenen Einzelinformationen eines Datensatzes dauerhaft speichern zu können, speichern wir sie in unserer eigenen Datenstruktur als JSON-Datei. Dabei gibt es eine globale Datei namens `schedule.json`, welche alle Informationen aller Tage beinhält. Ihr einziges Element ist eine Liste aller Tage. Jeder Tag ist folgendermaßen gegliedert:

```
{
    "date": {
        "day": 19,
        "month": 10,
        "year": 2015
    },
    "events": [...],
    "filename": "VplanLe19102015.xml"
}
```

Es ist ein Datum für den betreffenden Tags angegeben, der Dateiname für Nutzerinformation, und ein Array welches alle einzelnen Ereignisse `"change"` enthält. Jedes Event ist folgendermaßen aufgebaut.

```
{
    "change": "SUBJECT",
    "info": "Aufgaben im Unterrichtsraum",
    "new": {...},
    "old": {...},
    "room": "D309",
    "selector": {...},
    "targets": "6 6b Es Es Drm Drm",
    "time": "1"
}
```

Jedes Ereignis besitzt eine Information `"change"` darüber, welcher Änderungen vorgenommen wurden. Dabei gibt es vier verschiedene Änderungstypen mit jeweils spezifischen Kriterien.

Typ	Kriterien
`"SUBJECT"`	Unterrichtsfach wurde geändert
`"TEACHER"`	Unterrichtslehrer wurde gewechselt
`"ROOM"`	Raumwechsel
`"CANCELLED"`	Stunde fällt aus

Weiterhin beinhält ein Ereignis Information über die zwei betreffenden Stunden: die ursprüngliche Stunde `"old"` und die Vertretungsstunde `"new"`. Jede dieser Stunden hat die selbe Struktur:

```
{
    "subject": {
        "prefix": null,
        "subject": "De",
        "suffix": null
    },
    "teacher": {
        "full": "Frau Mittelstädt",
        "short": "Mi"
}
```

Eine Stunde enthält ein Fach `"subject"` und einen Fachlehrer `"teacher"`. Das Fach besteht wie schon oben beschrieben aus dem Fachnamen als Abkürzung, sowie einem optionalem Präfix und Suffix. Der Lehrer ist sowohl als Abkürzung `"short"` als auch als vollständiger Name `"full"` dargestellt.
Die betreffende Klasse bzw. Schülergruppe ist im Bezeichner `"selector"` gespeichert. Es gibt drei Typen von Selektoren, `"SIMPLE"`, `"MULT"` und `"COURSE"`, welche wir schon klassifiziert und mit jeweiligen Attributen versehen haben. Daher ist jeder Selektor aus den selben vier definierenden Attributen aufgebaut, sowie einer Information über den betreffenden Typ. Nutzt ein Typ ein Attribut nicht, so wird diesem der Wert `null` zugewiesen. Hier ein Beispiel für einen Selektor vom Typ `"SIMPLE"`:

```
{
    "grade": 6,
    "subclass": null,
    "subgrades": "b",
    "subject": null,
    "type": "SIMPLE"
}
```

Ein Ereignis beinhält weiterhin ein `"targets"`-Attribut, welches in einem String durch Leerzeichen separiert alle betreffenden Klassen und Lehrer für die Filter-Funktion beinhält. Natürlich wird auch die Zeit der Stunde gespeichert. Falls es sich um Mehrfach-Stunden handelt, so werden beide Ereignisse zu einem zusammengezogen, und die Zeitangabe wird durch ein mit - getrenntes Zeitintervall getrennt. Ist eine Textinformation vorhanden, so wird diese ebenfalls in `"info"` abgelegt.

3.3.7 Fehlerbehandlung und Logging

Der Verarbeitungsprozess besteht aus mehreren Schritten, die jeweils einzelne Fehlerbetrachtungen und ausführliche Logs haben. Jeder mögliche Fehler wird abgefangen, geloggt und ggf. automatisch behandelt. Dabei werden die Fehler in drei Log-Level unterteilt: `WARNING`, das niedrigste Level, welches einen Fehler beschreibt, der unerwünscht

ist, jedoch die Gesamtheit der Daten des Tages nicht gravierend beeinträchtigt, `ERROR`, ein Fehler, der das Abbrechen der Bearbeitung und somit den Fehlen des jew. Tages zur Folge hat, und `CRITICAL`, die höhste Fehlerstufe, welche nur komplett unerwartete Fehler die die gesamte Funktionalität der Applikation beeinträchtigen umfasst. Bei `CRITICAL`-Fehlern wird zudem automatischematisch eine Email an die System-Administratoren geschickt, und, wenn möglich, eine automatische Wiederherstellung des Systems eingeleitet.

3.4 Client

Der Client ist in diesem Fall ein Browser eines Besuchers der Seite. Das Gerät des Benutzers kann sowohl ein normaler PC als auch ein Smartphone oder ein Tablet sein. Aus Gründen der Sicherheit und Privatsphäre kann ein Server nicht vor der eigentlichen Kommunikation zwischen diesen Gerätetypen unterscheiden. Während es theoretisch möglich ist, nach dem Herstellen einer Verbindung über `AJaX` Daten nachzuladen ist es praktischer, wenn die Seite eigenständig bestimmen kann, um welchen Gerätetyp es sich handelt. Dementsprechend ist eine Reaktion und Anpassung des Seitenlayouts nach dem Übertragen der Daten durch das Ausführen von Code durch den Client notwendig.

Außerdem muss die Seite ohne Serverunterstützung sämtliche Aufgaben durchführen können, für die eine erneute Datenübertragung schlichtweg zu Zeitaufwändig wäre. Beispiele hierfür sind das Filtern nach bestimmten Lehrern bzw. Klassen, das Anzeigen von Menüs, das Wechseln zwischen verschiedenen Tagen etc.

Um eine Seite dynamisch und unabhängig zu machen, werden Stil- und Skriptsprachen verwendet. Die gebräuchlichen sind `CSS` und `JavaScript`.

3.5 Design

3.5.1 Tabellendarstellung

3.5.2 Slides

3.6 Interaktivität

3.6.1 Filter

3.6.2 Cookies

3.6.3 Menü

4 Benutzung