Georg-Christoph-Lichtenberg-
Oberstufengymnasium

Besondere Lernleistung Informatik

LegionBoard

Freies Vertretungsplansystem für Schulen

Autor:
Nico Alt
nicoalt@posteo.org

Betreuender Fachlehrer:
Sascha Richter

13. März 2017

Originalausgabe, 13. März 2017

Falls Sie eine digitale Version des Dokuments erhalten möchten, können Sie mir gerne eine E-Mail schreiben:
nicoalt@posteo.org

Inhaltsverzeichnis

1 Einleitung

Als ich in die Grundschule ging, hat das Internet im Alltag noch keine so große Rolle wie heutzutage gespielt. Smartphones hatten ihren großen Durchbruch erst Jahre später und so gab es unterwegs noch so gut wie keine Nutzung des Internets. Aus diesem Grund wurden damals Telefonketten genutzt, um die Eltern über den Ausfall eines Lehrers zu informieren.

Später wechselte ich auf eine weiterführende Schule und Telefonketten wären bei den dortigen Schülerzahlen und Ausfallquoten unmöglich gewesen. Deshalb erstellte man Vertretungspläne, die ausgedruckt und an mehreren Stellen in der Schule aufgehängt wurden. Das war damals, verglichen mit dem Konzept aus der Grundschule, ein großer Fortschritt.

Während der Zeit auf der weiterführenden Schule kam der große Durchbruch der Smartphones. Bald hatte jeder Schüler eines, ein Internetanschluss zu Hause und in der Schule war auch schon lange Standard, und so war die weitere Entwicklung vorhersehbar. Schon bald wurde der bisher ausgedruckte Vertretungsplan im Internet passwortgeschützt veröffentlicht und in der Schule auf einem zentralen Monitor dargestellt.

Dieser Sprung ins Internet ist zweifellos ein großer Schritt und eine Erleichterung im Alltag eines Schülers. Aber diese Lösung ist noch lange nicht ideal. Sie ist nicht wirklich sicher, die Darstellung auf mobilen Geräten ist alles andere als optimal und Benachrichtigungen bei Neuerungen sind nicht möglich. Außerdem sind alle bisherigen Vertretungsplansysteme proprietär (herstellergebunden) und die Schulen somit von den Lizenzen der Hersteller abhängig.

Zu Beginn meiner Qualifikationsphase dachte ich mir deshalb: Ich mach das jetzt mal anders.

1.1 Ausgangsbedingungen

Auf dem Oberstufengymnasium, das ich derzeit besuche, war man schon weiter als auf meiner bisherigen Schule. Dort hatte der ehemalige Schüler Tom Kurjak eine Webseite entwickelt, auf der die Schulleitung die Änderungen eintragen und die Schüler sie dann ansehen konnten. Diese grundlegende Idee benutzt auch LegionBoard.

Im Hintergrund griff ein *PHP*-Skript auf eine *MySQL*-Datenbank zu und stellte die Daten direkt im Browser dar. So konnte das Aussehen angepasst werden und es wurden bedeutend weniger Daten übertragen als bei der Lösung mit PDF-Dokumenten. Diesen Fortschritt finde ich wirklich bemerkenswert, da er die Technologien des Internets viel besser nutzt und besonders deshalb, weil er von einem Schüler entwickelt wurde. Das ermutigte mich, es selber auszuprobieren.

Allerdings war auch die Webseite von Tom nicht ideal. Sie war nicht für mobile Geräte optimiert, Benachrichtigungen waren nicht möglich und Einträge waren sehr primitiv, da sie nur aus dem Wochentag, dem Lehrer und einem Text bestanden.

Freundlicherweise stellte mir Tom den Quellcode seines Projekts zur Verfügung und so konnte ich mir die Funktionsweise genauer ansehen. Ich entschied mich dagegen, Teile seines Codes wiederzuverwenden, und beschloss, LegionBoard von Grund auf neuzuprogrammieren.

2 Konzept

Die Funktionsweise der Webseite von Tom ähnelte dem von mir entwickelten Alumni-Projekt, welches mein bis dahin umfangreichstes Projekt in *PHP* war. Ein Skript greift auf eine MySQL-Datenbank zu, verarbeitet die Daten und stellt das Ergebnis in Form von *HTML* dar. Diese Lösung hat jedoch, wie zuvor erwähnt, mehrere Nachteile. Aus diesem Grund sah ich mich nach Alternativen um.

Ich schaute mir an, wie die sogenannten „Big Player“ im Internet, also große Konzerne mit viel Erfahrung und Erfolg im Umgang mit dem Internet, ähnliche Probleme lösten. Dabei fand ich heraus, dass Google [1], Twitter [2], Facebook [3] und weitere Firmen dasselbe Prinzip benutzten. Bei vielen populären Diensten stand eine sogenannte *REST API* im Mittelpunkt.

2.1 REST API

Die *REST API* ist ein Nachfolge-Modell der *SOAP API*, wobei *REST* für für „representational state transfer“ steht, was auf Deutsch übersetzt „Repräsentativer Zustandstransfer“ bedeutet. Zwar wurde der Programmierstil *REST* schon im Jahr 2000 in der Dissertation von Roy Thomas Fielding [4] beschrieben, aber eine populäre Verwendung fand erst in den letzten Jahren statt.

Bei *REST* geht es im Grunde darum, sich verändernde Inhalte nach einem vorher festgelegten Schema zu veröffentlichen. Dabei ist es völlig offen, um welche Inhalte es sich handelt oder wie diese veröffentlicht werden. So war zum Beispiel auch die Webseite von Tom *REST*-konform, da sie den sich ändernden Inhalt (Ausfälle) nach einem festgelegtem Schema (Tabelle mit drei Spalten) veröffentlichte. Die meisten Nachrichten-Seiten dagegen sind nicht *REST*-konform, da sie die Nachrichten in unterschiedlicher Weise präsentieren.

Ob eine Webseite *REST*-konform ist oder nicht, kann man einfach feststellen: Auch wenn man keine Erfahrung im Programmieren hat, kann man sich überlegen, ob die Inhalte, die auf der Webseite veröffentlicht werden, einem festen Schema folgen, sodass zum Beispiel der Titel immer an der selben Stelle steht und auch ein Bild eine feste Position hat, sodass man beide

Informationen „blind“ auslesen kann.

Bei der Einheitlichkeit von *REST*-konformen Webseiten gibt es vier Prinzipien, die ich im folgenden erläutern werde. Das erste Prinzip sagt aus, dass alle *REST*-Anfragen sich selbst beschreiben sollen. Das wird durch die Nutzung der *HTTP*-Verben wie *GET*, *POST* und *DELETE* erreicht. Wenn später ein Schüler die Änderungen der aktuellen Woche angezeigt bekommen möchte, tut er das beispielsweise über eine *GET*-Anfrage. Im Gegensatz dazu erstellt die Schulleitung eine neue Änderung mit einer *POST*-Anfrage. Bei *SOAP*-konformen Seiten werden alle Anfragen über das *HTTP*-Verb *POST* gesendet, was dazu führt, dass eine weitere Information notwendig ist, damit der Server weiß, was der Client tun möchte.

Das nächste Prinzip beschreibt die Adressierbarkeit der Ressourcen. Wie zum Beispiel Häuser eine Adresse, bestehend aus Stadt, Straße und Hausnummer, haben, sollen auch die Ressourcen von LegionBoard (Änderungen, Lehrer, etc.) über eine Adresse erreichbar sein. Dabei „wohnen“ alle Ressourcen im gleichen Ort, was hierbei die Webseite des LOG ist. Dafür hat aber jede Ressourcengruppe eine eigene „Straße“, was im Falle von LegionBoard den Unterordner darstellt. Statt einer Hausnummer bekommt jede Ressource eine feste ID-Nummer zugewiesen.

Am Ende sieht es dann so aus, dass der Lehrer „Richter“ über die Adresse *log-web.de/legionboard/heart/v0/teachers/61* erreichbar ist. Dabei stellt *log-web.de/legionboard/heart/v0* den „Ort“ aller Ressourcen dar, *teachers* die „Straße“ der Ressourcengruppe „Lehrer“ und *61* die „Hausnummer“ des Lehrers „Richter“.

Das dritte Prinzip ist die Repräsentation der Ressourcen. Wie schon oben beschrieben, ist es für eine *REST*-konforme Webseite zwingend notwendig, die Art der Darstellung der Informationen nicht zu verändern. Die dafür genutzte Sprache wird dabei dem Entwickler überlassen. Während früher oft *XML* zur Überlieferung von Informationen benutzt wurde, hat sich seit einiger Zeit *JSON* als de-facto Standard durchgesetzt.

An dieser Stelle muss darauf hingewiesen werden, dass wir uns noch nicht mit den *API*s beschäftigt haben, sondern uns immer noch nur mit dem *REST*-Standard beschäftigen. Es ist deshalb auch legitim, *HTML* zur Darstellung der Daten zu benutzen, wie es etwa die Webseite von Tom getan

hat. Das ist besonders dann sinnvoll, wenn auf die Webseite nicht nur andere Computer Zugriff haben, sondern auch Menschen mit ihr täglich zu tun haben.

Das letzte Prinzip, das sowohl das wichtigste ist, als auch den sperrigsten Begriff hat, ist „Hypermedia as the Engine of Application State“, kurz *HATEOAS*. Dieses Prinzip sagt aus, dass alle Programme, die die Daten der Webseite verarbeiten, die API nur über *URL*s steuern, die vom Server festgelegt werden. Dieses Prinzip lockert die Abhängigkeit des Clients vom Server und ermöglicht es, die Schnittstelle zu ändern, während der Client gleich bleibt.

In der Realität sieht es jedoch so aus, dass die meisten Webseiten an diesem Punkt nicht *REST*-konform sind. Unter Entwicklern ist es umstritten, ob die mit diesem Prinzip eingeführte Komplexität ihren Nutzen wert ist, da die meisten der Meinung sind, dass bei einer Schnittstellenänderung auch der Client angepasst werden muss.

API steht für „application programming interface“, was auf Deutsch mit „Programmierschnittstelle“ übersetzt werden kann. Diese Programmierschnittstellen sind die Übergangspunkte zwischen zwei Programmen und meistens sehr gut dokumentiert.

Bei komplexen Systemen ist es sinnvoll, die verschiedenen Funktionen klar voneinander zu trennen, sodass zum Beispiel die Designer beim Ändern des Aussehens einer Webseite nicht die Logik zum Abrufen von Informationen aus der Datenbank stören können.

3 Realisierung

Da ich nun ein konkretes Konzept für die Umsetzung gefunden hatte, fing ich mit der Planung von LegionBoard an. Es war nun klar, dass im Mittelpunkt ein Server stehen sollte, der die Änderungen an die verschiedenen Clients verteilt. Der Hoster, auf dem die Webseite meiner Schule liegt, bietet nur *PHP* als Sprache und *MySQL* als Datenbank an, womit mir keine Wahl bei diesen Komponenten blieb. Ich hätte mich aber wahrscheinlich auch ohne diese Einschränkung für diese Kombination entschieden, da viele andere Software-Projekte ebenfalls erfolgreich darauf aufbauen (WordPress, Joomla, ...).

Da ohne diese Komponente bei LegionBoard nichts läuft und alle weiteren Komponenten davon abhängig sind, entschied ich mich, diese Software Heart, das englische Wort für „Herz“, zu nennen. Ich entschied mich wahrscheinlich deshalb für das Herz und gegen das Gehirn („Brain“), da wir gerade die Epoche der Romantik in der Schule behandelten. Wer weiß, wären wir noch bei der Aufklärung gewesen, sähe es heute vielleicht anders aus.

Auf Heart sollten dann die Clients zugreifen und die benötigten Daten anfordern können. Einer dieser Clients sollte eine Webseite werden, da fast alle Geräte mit Internetzugang über einen Browser verfügen und man somit mit einfachen Mitteln einen Großteil der Zielgruppe erreicht. Dass die Webseite auf *HTML* basieren sollte, war klar. Es stand lediglich zur Debatte, ob diese auf *JavaScript* oder *PHP* setzen soll. *JavaScript* wird von allen populären Browsern unterstützt und dank der enormen Rechenpower selbst kleinster Smartphones bietet *JavaScript* eine tolle Möglichkeit, die Server der Schulen zu entlasten, indem sie nur die Webseite ausliefern müssen und dann die Handys direkt auf Heart zugreifen. Außerdem ist es somit möglich, einen lokalen Client zu besitzen, da die Webseite ohne *PHP* statisch ist und der Browser somit nur *HTML* verarbeiten und *JavaScript* ausführen muss.

Nun fehlte nur noch ein Name für die Webseite. Im Prinzip ist es ja so, dass die Daten, die von Heart ausgegeben werden, für den Menschen schwer lesbar sind und erstmal für das *Auge* aufgehübscht werden müssen. Was liegt da näher, als die Webseite Eye zu nennen?

Mit Heart und Eye sind die Mindestanforderungen an ein laufendes Le-

gionBoard-System erfüllt. Heart verwaltet die Änderungen im Hintergrund und sowohl die Schulleitung, wie auch die Schülerschaft, greifen über Eye auf diese zu. Ein weiteres Desktop-Programm, exklusiv für die Schulleitung, ist nicht nötig und sogar noch hinderlich, da dann eine weitere Komponente für den Administrator hinzukommt, die er verwalten muss. Besteht das System aus Heart und Eye, muss er nur diese zwei Komponenten warten und jede beteiligte Person hat sofort die neuste Version zu Verfügung. Hier hilft das Motto: KISS - Keep it simple (and) stupid.

3.1 LegionBoard Heart

Heart steht im Mittelpunkt von LegionBoard und verwaltet alle Änderungen und daran beteiligte Ressourcen. Es ist in *PHP* geschrieben und greift mittels *mysqli* auf eine *MySQL*-Datenbank zu. Der Zugriff auf interne Bereiche wird mit Apache-Konfigurationen [5] verhindert, ebenso wie darüber die künstliche Weiterleitung der *HTTP*-Aufrufe zu Ressourcen realisiert wird.

3.1.1 Der Weg einer Anfrage

In Heart 0.2.0 gibt es vier Ressourcen: Änderungen (changes), Kurse (courses), Lehrer (teachers) und Aktivitäten (activities). Diese sind über *HTTP* abrufbar und greifen auf entsprechende *MySQL*-Datenbanken im Hintergrund zu. Die Ressourcen unterscheiden sich zwar in ihrem Inhalt, sind jedoch servertechnisch sehr ähnlich, da alle über die *HTTP*-Methoden *GET* (Auflistung), *POST* (Hinzufügung), *PUT* (Bearbeitung) und *DELETE* (Löschung) erreichbar sind. Außerdem sollen sie ausschließlich über ihren Namen (*GET /changes*) und nicht etwa über ihre Dateinamen (*GET /changes.php*) erreichbar sein. Aus diesen Gründen werden alle Anfragen, wie oben erwähnt, mit Hilfe von Apache auf eine zentrale Datei (*api.php*) umgeleitet.

Diese zentrale Einheit leitet alle Anfragen dann an die Klasse *LegionBoard* weiter und gibt Gzip-komprimiert die jeweiligen *JSON*-Daten aus. Dieser Schritt ist besonders für mobile Geräte wichtig, da die jeweiligen Anfragen an Heart bis zu fünf Mal größer sind, wenn sie nicht komprimiert werden. In Zeiten von bandbreitbeschränkten Mobilfunktarifen kann man sich diese Verschwendung nicht leisten.

In der Klasse *LegionBoard* wird die Anfrage dann auf die verschiedenen Ressourcen aufgeteilt. *LegionBoard* basiert auf der abstrakten Klasse *abstractAPI*, die sich um die Verarbeitung der Anfrage kümmert und viele grundlegende Aufgaben übernimmt. So werden dort die *HTTP*-Header ausgegeben, wie zum Beispiel *Content-Type: application/json*, was dem Client sagt, dass die Ausgabe der API ein *JSON*-Dokument ist. Außerdem wird dort die benutzte *HTTP*- Methode erkannt (*GET*, *POST*, ...), die zuständige Ressource festgelegt und die Zuweisung des Statustextes zu den jeweiligen Codes vorgenommen, wie zum Beispiel *404 Not found*.

Bevor die Anfrage von *LegionBoard* an den zuständigen Endpunkt weitergeleitet wird, werden die dafür benötigten Rechte überprüft. Jede Aktion, wie zum Beispiel „Änderung auflisten“ oder „Lehrer bearbeiten“, aber auch manche Felder, wie zum Beispiel der Grund von Änderungen, haben eine Indexnummer, die in der *MySQL*-Datenbank *Authentication* mit einem Benutzer verbunden ist.

Nachdem der Nutzer verifiziert und die Anfrage an den zuständigen Endpunkt weitergeleitet wurde, werden die Parameter aus der Anfrage ausgelesen und an die zuständige Ressource weitergeleitet. In der jeweiligen Ressource findet dann der Zugriff auf die Datenbank statt; vorher haben sich alle beteiligten Klassen nur um die Auswertung der *HTTP*-Anfrage gekümmert. Die Ressourcen führen dann je nach Anfrage einen *SELECT* aus (*GET*) oder erstellen (*POST*), bearbeiten (*PUT*) oder löschen (*DELETE*) ein Objekt.

3.1.2 HTTP-Statuscodes

Bei *GET*-Anfragen werden die Felder aus den *MySQL*-Datenbanken in ein multidimensionales Array geschrieben und an *LegionBoard* zurückgegeben. Wenn die Anfrage erfolgreich verläuft, wird der *HTTP*-Statuscode auf *200* gesetzt. Falls es einen Fehler in der Anfrage gibt, also zum Beispiel alle Änderungen eines Lehrers, der nicht existiert, angefordert werden, gibt es den Code *400* zurück. Wenn die Anfrage in Ordnung ist, aber keine Daten dazu existieren, weil zum Beispiel noch kein Kurs existiert, wird der Code *404* zurückgegeben.

Auch bei den restlichen Anfragen wird der Statuscode *400* zurückgegeben, wenn Fehler in den Parametern der Anfrage sind. Wenn die Anfragen erfolgreich verlaufen, gibt es bei *POST*-Anfragen den Statuscode *201* zurück, was für *Created* steht, und bei *PUT* und *DELETE* den Code *204* (*No Content*).

In den meisten Fällen sind das alle Statuscodes, die man von Heart erwarten kann. Man könnte auch sagen: Diese Codes werden erwartet und wenn man sie zurückbekommt, weiß man, was zu tun ist.

Leider kann es aber auch vorkommen, dass bei Ressourcen-verändernden Anfragen (*POST*, *PUT*, *DELETE*) der Statuscode *409* ausgegeben wird, der für den vielsagenden Statustext *Konflikt* steht. Dieser Statuscode wird ausgegeben, wenn der jeweilige *MySQL*-Befehl fehlschlägt und zum Beispiel

ein Kurs nicht erstellt werden kann, obwohl dies eigentlich möglich sein sollte. Um einen beliebten Kommentar in der Softwareentwicklung zu zitieren: „This should not happen.“ („Das sollte nicht passieren.“)

Wenn man diesen Statuscode zurückbekommt, kann man nicht viel mehr tun, als es erneut zu probieren oder den Administrator des Servers zu kontaktieren. Die Anzahl der möglichen Gründe, die diesen Status verursachen, ist schier unendlich. Es könnte beispielsweise sein, dass der *MySQL*-Server zum Zeitpunkt der Anfrage ein Update auf Systemebene durchläuft und deshalb nicht mehr ansprechbar ist. Eine andere mögliche Ursache ist, dass der verfügbare Arbeitsspeicher zu klein ist und der Task von *MySQL* vom unterliegenden Betriebssystem beendet wird. Die schlechte Nachricht ist: all diese möglichen Ursachen sind sehr schwer auf *PHP*-Ebene zu erkennen, weshalb keine genauere Nachricht als *Konflikt* ausgegeben wird. Aber die gute Nachricht ist: diese Fälle treten sehr selten auf, sind meistens einfach reproduzierbar und mit den Logs des Administrators leicht zu beheben.

3.1.3 Authentifizierung und Autorisierung der Anfragen

Jede Anfrage in Heart muss authentifiziert und autorisiert sein, damit sie weiterverarbeitet wird. Dies ist nötig, da die Daten, die mit LegionBoard ausgeliefert werden, verschiedenen Bestimmungen unterliegen und deshalb nicht öffentlich im Internet verteilt werden dürfen, sondern nur an eine „geschlossene Benutzergruppe“ von Schülern, Eltern und Lehrern [6].

Für die Authentifizierung der Anfrage gegenüber Heart wird ein SHA-256-Hash (Authentifizierungsschlüssel) der Login-Daten verwendet. Genauere Informationen dazu beschreibe ich im Unterunterabschnitt 3.2.4. Der Authentifizierungsschlüssel darf nur hexadezimale Zeichen (0-9, a-f) enthalten und ist damit aufgrund der Länge von 256 Bit immer 64 Zeichen lang. Generiert werden die Schlüssel von einer server-internen PHP-Seite, die manuell auf den Server hochgeladen werden muss [7]. Es ist somit nicht möglich, über die API Schlüssel zu erstellen oder zu bearbeiten.

In der Klasse *LegionBoard* wird der Schlüssel eingelesen und zusammen mit der gewünschten Funktion (Änderungen abrufen, Lehrer erstellen, etc.) an die Klasse *Authentication* weitergereicht. In dieser Klasse wird ein SHA-512-Hash des Authentifizierungsschlüssels erstellt, mit dem anschließend in

der *MySQL*-Datenbank überprüft wird, ob der Benutzer berechtigt ist, die von ihm angestoßene Funktion auszuführen.

Dadurch, dass nicht der Authentifizierungsschlüssel in der Datenbank gespeichert ist, sondern dessen SHA-512-Hash, sind die Zugangsdaten im Fall einer Datenbanklücke gesichert und können nach einem Angriff nicht verwendet werden.

Da der Schlüssel aber bei jeder Anfrage an Heart mitgesendet wird, ist es möglich, bei unverschlüsselten Verbindungen (*HTTP*) einen sogenannten „Replay-Angriff“ auszuführen, bei dem die Datenübertragung mitgeschnitten wird und die ermittelten Daten zu einem erneuten Ausführen der Funktion genutzt werden.

Ich habe mich dagegen entschieden, Lösungen zu implementieren, die diese Art der Replay-Angriffe verhindern, und empfehle stattdessen jedem Betreiber einer LegionBoard-Instanz ausdrücklich, mindestens Heart nur über *HTTPS* erreichbar zu machen. Damit wird die Möglichkeit eines Replay-Angriffs oder ein Mitschneiden des Authentifizierungsschlüssels und die Gefahr eines Man-in-the-Middle-Angriffs ausgeschlossen, beziehungsweise stark reduziert.

Erst wenn eine Anfrage diesen Authentifizierungsprozess durchlaufen hat, wird die eigentliche Funktion ausgeführt.

Wird einer Anfrage kein Authentifizierungsschlüssel angefügt, wird der *HTTP*-Status „401: Unauthorized“ zurückgegeben. Aber auch wenn der Benutzer nicht zum Ausführen der Funktion berechtigt ist oder gar nicht existiert, wird dieser Status zurückgegeben.

3.2 LegionBoard Eye

Eye ist die zentrale Schnittstelle in LegionBoard und ermöglicht sowohl den Schülern, die Änderungen anzusehen, als auch der Schulleitung und anderen berechtigten Personen, Änderungen und weitere Ressourcen zu verwalten. Im Gegensatz zu Heart ist es nicht in *PHP*, sondern in *JavaScript* geschrieben. Die Gründe für diese Wahl habe ich schon im Abschnitt 3 erläutert.

LegionBoard Eye greift auch nicht direkt auf die *MySQL*-Datenbank zu, sondern über *HTTP* auf die *REST API* von Heart. Doch bevor Eye eine Anfrage an Heart tätigt, muss sich der Benutzer erst authentifizieren. Auf dieses Thema gehe ich im Unterunterabschnitt 3.2.4 ein.

Einen grundlegenden Unterschied gibt es noch zwischen Eye und Heart: Während in einem LegionBoard-System nur eine Instanz von Heart existieren kann, ist es theoretisch möglich, dass sich jeder Schüler eine eigene Instanz von Eye einrichtet. (Eigentlich ist das auch der Standardfall, da der *JavaScript*-Code, im Gegensatz zum *PHP*-Code, auf vielen verschiedenen Computern heruntergeladen und ausgeführt wird. Da dies aber unbewusst geschieht, kann man im Grunde von „einer“ Instanz sprechen.)

3.2.1 Benutzeroberfläche

Der mit Abstand aufwändigste Teil der Entwicklung von LegionBoard war die Benutzeroberfläche. Während ich Heart vergleichsweise flott entwickeln konnte, hat Eye ziemlich viel Zeit verschlungen, was zu einem großen Teil daran lag, dass ich mich in der Entwicklung von Backends besser auskenne und auch schon ein Projekt mit *PHP* entwickelt hatte, aber noch keines mit *JavaScript*.

Bevor ich anfing, die Funktionen in Eye zu implementieren, machte ich mich erstmal auf die Suche nach einem Framework für die Benutzeroberfläche. Meine Ziele waren dabei, dass das Framework gut unterstützt, unter einer freien Lizenz veröffentlicht, aktiv weiterentwickelt und außerdem am Ende wenig zusätzliche Bandbreite kosten sollte.

Die ersten Versuche startete ich mit Skeleton [8], da ich dieses schon auf meiner Webseite einsetzte [9], doch ich merkte schnell, dass es zu simpel für die Ansprüche von LegionBoard gehalten war und ich trotzdem noch viele

Arbeiten zu erledigen gehabt hätte, die eigentlich ein Framework abnehmen sollte.

Nach kurzer Zeit wechselte ich deshalb zu Bootstrap [10]: Im Gegensatz zu Skeleton ist es zwar nicht so leichtgewichtig, dafür ist aber die Unterstützung von Drittbibliotheken besser und die Community ist um einiges größer, weshalb es auch aktiver weiterentwickelt wird.

Zusätzlich zu Bootstrap sind dann noch zwei Plugins in Eye aktiv: „Bootstrap Dropdown Menus Enhancement“ [11], um Checkboxen in den Menüs zu haben (zum Beispiel beim Filtern nach Lehrern), und „Bootstrap Datepicker“ [12], um einen Kalender für das Datum anzeigen zu lassen.

Neben Bootstrap und dessen Plugins gibt es noch drei weitere Bibliotheken: „jQuery“ [13], das das Programmieren mit *JavaScript* und die Kompatibilität mit den verschiedenen Browsern stark vereinfacht, SweetAlert [14], was die Meldungsfenster verschönert, und „jsSHA“, das das Hashen der Zugangsdaten übernimmt.

3.2.2 Gebündelter versus doppelter Code

In der Welt des Programmierens gibt es mehrere Regeln, an die sich alle guten Entwickler halten. Eine davon ist, Code niemals zweimal zu schreiben.

Wie schon in Heart angesprochen, habe ich während der Entwicklung darauf geachtet, die Bandbreite der Nutzer nicht unnötig zu belasten. Dort habe ich dafür gesorgt, dass die Ausgaben der API Gzip-komprimiert sind. Da der gesamte Code in Heart in *PHP* geschrieben ist, wird dieser auch nur auf dem Server aufgeführt und muss beziehungsweise darf deshalb nicht auf die Geräte der Nutzer heruntergeladen werden.

Bei Eye hingegen muss der gesamte Code, der für die Darstellung der aktuellen Seite benötigt wird, auf das Gerät heruntergeladen werden, da er in *JavaScript* geschrieben ist und deshalb ausschließlich auf dem Gerät ausgeführt wird.

In der Welt der Netzwerke gibt es dagegen auch ein paar Regeln, an die sich alle guten Webseiten halten. Eine davon ist, so wenig wie möglich Anfragen zu starten. Das liegt daran, dass es bei jeder neuen Anfrage einen Overhead gibt, der die Gesamtzeit verlängert.

Ein Beispiel ist das Herunterladen einer Datei. Während eine ein Gigabyte

große Datei ziemlich schnell heruntergeladen ist, sieht es bei einer Million kleinen Dateien mit der Gesamtgröße von einem Gigabyte schon ganz anders aus. Für jede einzelne Datei muss die Anfrage neu „ausgehandelt“ werden, was bedeutet, dass der Benutzer eine Anfrage an den Server schicken muss, dieser sie verarbeitet und dann die Datei mit zusätzlichem Overhead zurückschickt. Obwohl sich die Gesamtgröße der beiden Daten-Pakete nicht unterscheidet, wird der zweite Download erheblich länger brauchen.

Bei Eye habe ich es deshalb vermieden, Code in andere Klassen auszulagern, und habe die meisten Funktionen in den Klassen der entsprechenden Funktionen belassen. Nur einzelne Funktionen habe ich in einer eigenen Klasse namens „Utilities“ zusammengetragen, aber schon dabei gibt es einigen Code, der meistens nicht gebraucht wird.

Eine Lösung des Problems wäre es, den *JavaScript*-Code vor dem Hochladen auf den Server zusammenzubauen, um so nur eine einzige Anfrage zu benötigen. Das erschwert jedoch den Veröffentlichungsprozess, da dann nicht einfach nur der Quellcode heruntergeladen, entpackt und direkt wieder auf den Server hochgeladen werden kann.

Eine andere, partielle Lösung des Problems, die schon heute in Eye eingesetzt wird, ist das Zwischenspeichern (caching). In entsprechenden Apache-Konfigurationen wird dem Server mitgeteilt, dass einzelne Dateitypen vom Gerät zwischengespeichert werden können [15]. Damit lädt der Browser eine Datei nur einmal vom Server herunter. Bei der nächsten Benutzung lädt er sie dann aus dem Zwischenspeicher, statt eine neue Anfrage zu starten.

Aus letzterem Grund werde ich mich wahrscheinlich bei der Weiterenwicklung von Eye mehr an die erste Regel gegen doppelten Code halten und die Funktionen objektorientiert auf mehrere Klassen verteilen.

3.2.3 Unterstützung verschiedener Browser

Obwohl wir mittlerweile im Jahr 2017 sind, werden noch Betriebssysteme aus dem Jahr 2001 (Windows XP) und Browser aus dem Jahr 2005 (Internet Explorer 7) eingesetzt. Das heißt, dass damit mindestens zwölf Jahre Weiterenwicklung und Absicherung einfach ausgelassen werden. Damit man einen Vergleich hat: Erst zwei Jahre nach der Veröffentlichung des Internet Explorers hat Apple das erste iPhone vorgestellt und ist mittlerweile bei Version

7 des Geräts. Außerdem ist der 2004 veröffentlichte Browser Firefox mittlerweile bei Version 50+ angelangt und unterstützt immer noch Windows XP.

Wie man sieht, kann ich eine Nutzung uralter Browser-Versionen nicht nachvollziehen, ungeachtet der damit einhergehenden Sicherheitslücken.

Die Nutzung von LegionBoard Eye ist daher in alten Browsern eingeschränkt und nicht unterstützt. So funktioniert in dem eben erwähnten Internet Explorer 7 zwar das Abrufen von Änderungen, aber jegliches Erstellen, Bearbeiten und Löschen von Ressourcen schlägt fehl. Da ich bisher keine ausgiebigen Tests gemacht habe, gibt es auch keine offiziellen Mindestanforderungen für Eye.

In der Praxis sieht es aber zumindest so aus, dass die Webseite auf den meisten Browsern eingesetzt werden kann, die grundlegende Funktionen von HTML 5 unterstützen. Bei Firefox sind dies alle Versionen ab 3.5, bei Chrome geht es mit der Version 3.0 los und der Internet Explorer kann ab der Version 9 eingesetzt werden.

Der Einsatz derart alter Browser kommt jedoch nur recht selten vor und kann durch die Installation von Firefox schnell gelöst werden.

Leider gab es aber auch in der Vergangenheit einzelne Fälle, bei denen der Login in Safari auf verschieden iOS- und macOS-Geräten nicht funktioniert hat. Diese Fälle waren auf verschiedene Versionen verteilt und konnten auf anderen Geräten zuerst nicht reproduziert werden. Zwar konnten auch diese Fälle durch die Installation von Firefox beziehungsweise Chrome gelöst werden, doch es wäre trotzdem wünschenswert, die Ursache des Fehlers herauszufinden und zu beheben. Mittlerweile ist das Problem glücklicherweise gelöst. Details dazu befinden sich im Unterabschnitt 6.1.

3.2.4 Authentifizierung und Autorisierung von Benutzern

Wie im Unterunterabschnitt 3.1.3 beschrieben, muss jede Anfrage in Heart authentifiziert und autorisiert sein, bevor sie weiter verarbeitet wird.

Aus diesem Grund ist das erste, was einem neuen Nutzer in Eye angezeigt wird, der Login-Dialog. Wenn der Benutzer seine Zugangsdaten eingegeben hat, wird dessen SHA-256-Hash generiert und auf seinem Computer gespeichert, sodass er bei einem erneuten Aufruf angemeldet bleibt. Dadurch, dass

nur der Hash der Zugangsdaten gespeichert ist, sind die Passwörter der Nutzer gesichert und können somit nicht zu einer Account-Übernahme bei anderen Anbietern führen. In Version 0.2.0 wird der Hash im sogenannten *HTML5 Web Storage* [16] abgespeichert. Wenn diese Funktion nicht zur Verfügung steht, weil der Browser beispielsweise zu alt ist, wird ein Cookie mit einer Laufzeit von 3 Jahren (durchschnittliche Verweildauer auf Oberstufengymnasien) erstellt. In Zukunft ist es geplant, nur noch den *Web Storage* zu benutzen, da mittlerweile schon Generationen von Browsern die Technik unterstützen und Eye bei Browsern, die das nicht tun, sowieso nur eingeschränkt läuft.

Durch diese Umstellung kommt Eye in den Genuss des sogenannten *sessionStorage*. Damit ist es möglich, Daten nur solange zu speichern, wie die *Session* des Benutzers dauert. Bei den meisten Browsern ist das in der Regel die Lebensdauer des Tabs. Wenn der Tab mit der Seite geschlossen wird, werden die Daten, die in dem *sessionStorage* gespeichert waren, gelöscht.

Das ist insofern sinnvoll, dass sich der Nutzer bei jeder neuen Nutzung wieder neu authentifizieren muss. In Version 0.2.0 ist der Nutzer solange eingeloggt, bis er sich manuell ausloggt. Auf öffentlichen Rechnern, die von mehreren Personen benutzt werden, ist das ein Sicherheitsrisiko, da sensitive Daten von Dritten eingesehen werden könnten, wenn sich der Nutzer nicht manuell abmeldet.

LegionBoard Eye versucht erst, eine Anfrage an Heart zu machen, wenn der Nutzer dessen Zugangsdaten eingegeben hat. Es ist insofern unmöglich, Eye ohne Passwortschutz zu betreiben, zumal Heart bei leerem Authentifizierungsschlüssel immer den *HTTP*-Status „401: Unauthorized“ zurückgibt.

Wenn der Nutzer einmal seine Zugangsdaten eingegeben und Eye den Authentifizierungsschlüssel generiert hat, wird dieser jedes mal als Parameter „k“ (key) an die Anfrage angehängt. Heart übernimmt dann die Authentifizierung und Autorisierung und meldet entweder das Ergebnis der Anfrage oder den *HTTP*-Status „401: Unauthorized“ zurück.

Eye überprüft nicht, welche Rechte der Benutzer hat, sondern sendet jede Anfrage nach dem Prinzip „Trial and error“ (Versuch und Irrtum). Wenn der Nutzer zu einer Aktion nicht berechtigt ist, meldet Heart dies zurück und der Nutzer wird darauf hingewiesen.

Dieses Verhalten hat sowohl bei der Schülerschaft als auch bei der Schulleitung Verwirrung ausgelöst, weshalb geplant ist, am Anfang jeder Session die Berechtigungen eines Nutzers abzufragen und nur entsprechende Aktionen anzuzeigen [17].

3.3 LegionBoard KISS

Erst nachdem die Entwicklung von Eye zum größten Teil abgeschlossen war und die Veröffentlichung von Version *0.1.0* kurz bevorstand, habe ich bemerkt, wie unglaublich komplex der Code geworden ist. Ich hatte zwar zu Beginn ein grobes Konzept, wie der Code von Eye aussehen soll, doch da es mein erstes Projekt dieser Art war, kamen während der Entwicklung immer wieder neue Anforderungen und Probleme auf, mit denen ich vorher nicht gerechnet hatte. So kam es dann bald dazu, dass ich Eye in immer kleinere Meilensteine unterteilte und von Feature zu Feature entwickelte, ohne dabei den Umfang des Codes im Auge zu behalten. Natürlich achtete ich darauf, objektorientiert zu programmieren, keinen Code doppelt zu schreiben und möglichst wenige Abhängigkeiten zu anderen Frameworks zu bauen, um den Code nicht unnötig aufzublähen. Aber meine Vorstellungen vom Umfang von Eye und dessen späterer tatsächlicher Umfang gehen weit auseinander.

Später habe ich einiges an Arbeit darin gesteckt, den Code zu vereinfachen und ihn damit besser instand halten zu können. Aber es ist trotzdem nicht möglich, ein solch komplexes Programm mit all seinen Möglichkeiten in wenige Zeilen Code zu verpacken. Wenn man sich eingearbeitet oder schon an ähnlichen Projekten gearbeitet hat, ist der Code überschaubar. Für Personen, die noch nicht viel Erfahrung in der Entwicklung komplexer Software haben, sind die Einstiegshürden aber immer noch hoch.

Auf der Webseite LegionBoards steht: „Der Quelltext wurde veröffentlicht, damit jede Schule rund um den Globus es frei benutzen, studieren und modifizieren kann.“ [18]

Das stimmt auch so. Der komplette Quellcode von LegionBoard ist frei und öffentlich einsehbar, man kann mit Hilfe der Installationsanleitungen [19] innerhalb kurzer Zeit LegionBoard installieren und Modifikationen an der Software sind erlaubt. Allerdings sind die Einstiegshürden wie oben genannt hoch und eine tiefe Auseinandersetzung mit dem Quellcode ist eher unwahrscheinlich. Um diese Hürde zu vermindern, entschloss ich, einen einfachen Client zu bauen: „KISS - Keep it simple and stupid.“

3.3.1 Realisierung

Das Ziel von KISS ist es, die Änderungen von Heart mit möglichst wenig absoluten Zeilen Code benutzerfreundlich darzustellen. Ein Framework wie Bootstrap, das bei Eye verwendet wird, ist sehr umfangreich und bietet viele Funktionen, die für eine einfache Darstellung nicht nötig sind. Ich entschied mich, kein Framework, noch nicht einmal ein *CSS*-Stylesheet zu verwenden, sondern stattdessen auf das gute, alte *HTML*-Standardlayout zu setzen.

KISS ist auch nicht für die Massennutzung gedacht. Wie oben erwähnt, soll es ein einfacher Einstieg für Interessierte und somit ein „Client für Hacker“ sein. Aus diesem Grund ist die Parametereingabe auch so minimalistisch wie möglich: über *GET*-Parameter in der *URL*. Ein Formular würde sowohl das *HTML*-Dokument wie das *JavaScript*-Programm nur unnötig aufblähen. Die Abfrage der *URL*-Parameter dagegen ist nur eine kleine, zehnzeilige Funktion, die am Ende des *JavaScript*-Programms angehängt ist. KISS ist auf der Webseite verfügbar [20] und man kann jede Schule mit aktueller Heart-Version aufrufen. Eine Alternative zu den *GET*-Parametern mit weniger Zeilen Code gibt es meiner Meinung nach nicht.

Die größte Hürde in der Entwicklung von KISS war es, die *HTTP*-Anfragen mit *JavaScript* zu bauen, da ich bei Eye *jQuery* verwendet hatte. Doch da KISS ja nur die Änderungen auflisten sollte, wurden nur *GET*-Anfragen benötigt und ich fand schon nach kurzer Recherche eine Lösung.

Nachdem alle benötigten Ressourcen angekommen sind, sortiert KISS die Änderungen nach dem Startdatum, ersetzt *ID*s und Indexnummern durch Namen und Begriffe, formatiert die Start- und Endzeiten und fügt alles zu einer Liste zusammen.

Am Ende kann man mit ungefähr 150 Zeilen Code die Änderungen seiner Schule abrufen, was eine enorme Vereinfachung zu den ungefähr 2000 Zeilen Code von LegionBoard Eye darstellt.

4 Einsatz

Die beste Software bringt nichts, wenn sie nicht eingesetzt wird. Deshalb achtete ich schon während der Entwicklung darauf, dass LegionBoard möglichst einfach und schnell eingerichtet werden kann und die Einstiegshürden so niedrig wie möglich sind.

Bei Heart sieht das so aus, dass man beim Start folgende Dinge konfigurieren muss: einmal die *MySQL*-Daten, bestehend aus Host, Benutzer, Passwort und Datenbank, und dann noch die Benutzer, die später auf LegionBoard zugreifen dürfen (mit Hilfe einer Webseite im Browser).

Während bei Eye die Konfiguration nur aus dem Eintragen der *URL* von Heart besteht, muss KISS gar nicht eingerichtet werden. Dort werden alle benötigten Daten - Benutzer, Passwort und API - mit jeder Anfrage als URL-Parameter übertragen.

Sowohl bei Heart als auch bei Eye gibt es Installationsanleitungen [19], welche die eben kurz zusammengefassten Einrichtungsschritte nochmal ausführlich erläutern.

4.1 Umstellung im LOG

Die Umstellung an meiner Schule, dem Lichtenberg-Oberstufengymnasium in Bruchköbel, lief reibungslos ab, was auch an dem tollen Support der Webmaster lag.

Konkret ist die Umstellung in drei Teile unterteilt: zuerst die Einrichtung von LegionBoard Heart und Eye in einem separatem Ordner (*/legionboard/heart*; etc.), anschließend die Weiterleitung vom alten System auf das neue, und zu guter Letzt die Einweisung aller beteiligten Personen, inklusive Sekretariat und Schülerschaft.

Zum Zeitpunkt der Einrichtung am LOG lagen noch keine Installationsanleitungen [19] vor, doch die Einrichtung verlief genau wie dort beschrieben ab. Ich kontaktierte den für die Webseite unserer Schule zuständigen Webmaster und bat ihn, den Ordner „/legionboard“ und die *MySQL*-Datenbank „legionboard“ einzurichten und mir per *FTP* Zugriff auf den Ordner zu geben. Ich erstellte dann die Ordner „heart“ und „eye“ und lud dort die jeweils benötigten Dateien hoch.

Eine Besonderheit gibt es jedoch: die API von Heart liegt in dem Unterverzeichnis „v0“, für den Fall, dass später verschiedene Versionen der API parallel laufen sollen.

Nachdem alles hochgeladen war, trug ich die benötigten Daten in die Konfigurationsdateien von Heart und Eye ein und bereitete das Benutzererstellungswerkzeug von Heart vor. Später in der Schule erstellte dann die Sekretärin zusammen mit mir die Accounts für die Schulleitung und die Schülerschaft. Anschließend zeigte ich ihr LegionBoard und wir erstellten ein paar Änderungen und Lehrer.

Bis zum nächsten Schritt - der Weiterleitung vom alten System auf's neue - lag es nun an der Schulleitung, die Umstellung vorzubereiten. Es mussten nun alle Lehrer und Änderungen, startend ab dem geplanten Umstellungsdatum, eingetragen werden. Eine Übernahme der Daten aus dem alten System fand nicht statt, da einerseits die vergangenen Änderungen nicht benötigt wurden und andererseits die Lehrer nicht wie bei LegionBoard in separaten Datensätzen gespeichert wurden und sich ein Import deshalb erschwerte.

Während der Entwicklung beachtete ich diese Masseneintragung zu Beginn der Nutzung und gestaltete die Hinzufügen-Seiten benutzerfreundlich. Wenn man einen Lehrer einträgt, wird automatisch das Namensfeld fokussiert, und bei einem Tippen der Enter-Taste wird der Lehrer erstellt. Mit einem weiteren Enter schließt man dann den Bestätigungsdialog und das Namensfeld wird automatisch geleert und erneut fokussiert, sodass man während dem massenhaften Erstellen von Lehrern oder Kursen ausschließlich die Tastatur benutzen und die Maus links (oder rechts) liegen lassen kann.

Die Schulleitung und ich planten, LegionBoard nach den Osterferien einzusetzen. Aus diesem Grund konnte ich nicht am letzten Freitag vor den Ferien nach der dritten Stunde, wie sonst üblich, in die Ferien starten, sondern musste noch auf einen kurzen Besuch bei der Schulleitung vorbeischauen. Zusammen mit dem Webmaster deaktivierte ich die bisherige Zugangsprüfung per Apache und richtete eine Weiterleitung auf LegionBoard ein.

Hier ist ein Negativpunkt der Umstellung: die Benutzer konnten sich nicht mehr auf ihren im Browser integrierten Passwort-Manager verlassen, sondern mussten beim erstmaligen Anmelden in LegionBoard die Zugangsdaten erneut eingeben. Da die meisten Personen diese aber das letzte Mal vor über

einem Jahr eingegeben hatten und das menschliche Gehirn im Merken von (einmalig benötigten) Passwörtern nicht sehr gut ist, führte das dazu, dass mich während der Osterferien mehrere Personen kontaktierten, um die Zugangsdaten des Vertretungsplans zu erfahren.

Rückblickend gesehen ist eine solche Umstellung wahrscheinlich der beste Zeitpunkt, um eine Aktualisierung der Zugangsdaten vorzunehmen. Doch daran haben wir zu dem Zeitpunkt nicht gedacht.

4.2 Kontakt mit weiteren Schulen

Um LegionBoard anderen Schulen bekannt zu machen, habe ich nach der Veröffentlichung von LegionBoard *0.1.0* eine Webseite [23] erstellt, auf der über LegionBoard, seine Komponenten und die Vorteile zu anderen Vertretungsplansystemen informiert wird. Ich habe daraufhin Schulen in meiner Region angeschrieben, dabei LegionBoard vorgestellt und einen Einsatz an ihrer Schule angeboten, doch bis heute ist das LOG die einzige Schule, die LegionBoard einsetzt.

5 Quellcodeveröffentlichung

Schon bevor ich die ersten Zeilen Code geschrieben hatte, war für mich klar, dass ich den Quellcode von LegionBoard unter einer freien Lizenz veröffentlichen würde. Damit würde ich nicht nur jedermann ermöglichen, den Code zu modifizieren und auf Sicherheitslücken zu überprüfen, sondern auch die Schulen von der bisherigen Abhängigkeit proprietärer Software und den damit verbundenen Lizenkosten befreien. Das Ziel für LegionBoard war und ist es auch heute noch, eine Software zu bauen, mit deren Hilfe Schulen unabhängig von Dritten eine langfristige Lösung für ihren Vertretungsplan haben. In meinen Augen ist das nur durch freie Software möglich.

In der Vergangenheit habe ich schon an diversen Open-Source-Projekten mitgearbeitet und auch schon selber Software-Projekte ins Leben gerufen. So ist HA-Manager [21], eine Android-App zum Verwalten von Hausaufgaben und eine meiner ersten Apps, ebenfalls seit zwei Jahren Open Source. Mit der Quellcodeveröffentlichung hat die Entwicklung von HA-Manager einen derartigen Schub bekommen, mit dem ich vorher nicht gerechnet hatte. Kurze Zeit nach besagtem Ereignis bekam ich schon erste Pull Requests (Weiterentwicklungen des Codes mit Bitte um Aufnahme in Hauptprojekt) von der anderen Seite des Globus, zahlreiche fremde Leute reichten Verbesserungsvorschläge ein und diskutierten an bestehenden Problemen mit. Heute ist HA-Manager in elf Sprachen übersetzt [22], darunter Französisch, Polnisch und Arabisch, und noch heute kommen weitere Übersetzungen hinzu. Ohne Quellcodeveröffentlichung wäre das alles nicht möglich gewesen.

Bei meinen bisherigen Projekten habe ich zwei verschiedene Lizenzen benutzt: die MIT-Lizenz [24] und die „General Public License“, kurz GPL [25]. Ich weise an dieser Stelle darauf hin, dass ich kein ausgebildeter Jurist bin und deshalb nur einen groben Überblick über die Lizenzen geben kann.

Die MIT-Lizenz ist im Gegensatz zur GPL sehr einfach gehalten und sagt im Grunde eines aus: Mach mit dem Code, was Du willst, aber erwähne dabei den Autor, von dem Du ihn hast.

Die GPL kann man dagegen unmöglich in einem Satz vollständig zusammenfassen. Im Vergleich zur MIT-Lizenz ist sie ein gewaltiges Monster von großem Umfang und selbst Mitglieder der Free Software Foundation

(FSF) [26], die quasi die „Mutter“ der Lizenz ist, streiten sich um die Interpretation einzelner Abschnitte. Doch die Lizenz bietet einige Sicherheiten, die die MIT-Lizenz nicht zusichert: so muss nicht nur der Autor angegeben werden, sondern der modifizierte Quellcode muss ebenfalls wieder unter der GPL veröffentlicht werden. Das beugt der Situation vor, in der Personen oder Firmen den Quellcode verwenden, ohne ihn anschließend wieder zu veröffentlichen. Zudem hat die GPL in manchen Open Source-Kreisen eine höhere Akzeptanz; so habe ich zum Beispiel schon gehört, dass jemand keinen Beitrag zu einer Software leisten wollte, weil diese nicht unter der GPL veröffentlicht war und somit Firmen den teilweise hart erarbeiteten Code „klauen“ konnten.

Eine weitere wichtige Entscheidung bei einem Open Source-Projekt ist die Wahl des Versionierungssystems und des Hosters. Seit einigen Jahren ist Git [27] der absolute Standard. Es hat viele Vorteile gegenüber älteren Systemen wie Subversion und eine große Unterstützung seitens der Community.

Bei der Wahl des Hosters ist die Situation nicht so eindeutig: Während die meisten auf den (ehemaligen) Fast-Standard GitHub [28] setzen, meinen die anderen, dass quelloffene Software auch von ebenso quelloffener Software gehostet werden sollte und setzen deshalb auf GitLab [29]. Außerdem gibt es noch die Fraktion der Dezentralen, die Software nur auf eigenen, in letzter Zeit immer öfters mit GitLab betriebenen Servern hostet. Im Idealfall würde ich wahrscheinlich auch letzterer Gruppe angehören, aber in der Wirklichkeit ziehe ich eine zentrale, kostenlose Instanz auf GitLab.com vor, da damit erstens keine Kosten meinerseits für einen Server anfallen und zweitens sich Benutzer nicht extra auf meiner eigenen Plattform anmelden müssen, sondern mit einem Account alle Projekte auf GitLab.com erreichen können. Außerdem ist die verwendete GitLab-Version immer up-to-date und ich muss mich nicht um die Einrichtung und Wartung kümmern.

GitLab zog ich deswegen GitHub vor, weil es quelloffen ist und schon damals eine bessere Benutzungserfahrung und größeren Funktionsumfang besaß. Das einzige Argument, bei dem GitHub im Moment noch trumpfen kann, ist die Anzahl der Nutzer. GitHub hatte lange Zeit eine Monopolstellung in der Software-Hosting-Szene und deshalb besitzen viele Entwickler und fortgeschrittene Nutzer einen Account auf dieser Plattform, doch neben diesem

Argument bleibt nicht viel übrig. Selbst wenn es GitHub schaffen sollte, einige Features aus GitLab nachzubauen, wird GitLab immer einen Vorteil haben, da es quelloffen ist und somit jedermann daran arbeiten kann.

Da ich trotzdem nicht die große Nutzerzahl GitHubs verlieren wollte, entschied ich mich, das Projekt dort zu spiegeln. Das heißt, dass alle organisatorischen Dinge, wie Issues und große Merge Requests, auf GitLab ablaufen, aber man das Projekt auch auf GitHub forken und Pull Requests stellen kann. Mit dieser Methode verbindet man die Stärken GitLabs mit der hohen Nutzerzahl GitHubs.

5.1 Kompatibilität zu fremden Programmen

Es war von Beginn an mein Wunsch, dass LegionBoard von vielen Drittprogrammen unterstützt wird. So dokumentierte ich Hearts API [30] während der Entwicklung ausführlich und achte auch bei Weiterentwicklungen darauf, dass die Dokumentation auf dem neusten Stand ist. Aus eigener Erfahrung weiß ich, wie wichtig eine gute Dokumentation ist, da es den Einstieg in ein Projekt sehr erleichtert oder auch überhaupt erst ermöglicht.

Parallel dazu kommentierte ich den Code an vielen Stellen und erläuterte dabei, was der Code an dieser Stelle bewirkt. So muss man sich auch nach längerer Zeit nicht lange in den Code einarbeiten, sondern kann direkt mit dem Programmieren anfangen.

Nach der Quellcodeveröffentlichung fing ich an, erfolgreiche Software in diesem Bereich zu suchen und fand dabei die Vertretungsplan-App von Johan von Forstner.

5.1.1 Substitution-Parser-Library

Johan hatte wie ich in seiner Schulzeit eine Android-App für seine damalige Schule entwickelt und ist dabei noch einen Schritt weitergegangen als ich: Statt den Vertretungsplan einfach nur herunterzuladen und darzustellen, schrieb er eine Server-Software, die den Vertretungsplan einlas und dann in maschinenlesbarer Form an die Android-App weitergab. Die App holte sich dann statt des Vertretungsplans die Daten des Servers und stellte den Inhalt mit Android-Boardmitteln dar. So war es Johan möglich, das Aus-

sehen perfekt an Mobilgeräte und Tablets anzupassen, den Vertretungsplan auch in Widgets darzustellen und die Nutzer bei Aktualisierungen des Vertretungsplans zu informieren.

Er merkte, dass diese Funktionalität auch für andere Schulen interessant war und arbeitete deshalb an einer Version des Servers und der App, die auch andere Schulen unterstützte. Den Quellcode der Software veröffentlichte er anschließend unter der GPL-Lizenz auf GitHub [31].

Der Erfolg gab Johan Recht: Innerhalb kurzer Zeit unterstützte die App unzählige Schulen und ist heute bei einem Stand von mehr als 430 Schulen angelangt.

Als ich Johan das erste Mal wegen einer Unterstützung für LegionBoard anschrieb, hatte er aber schon eine Entscheidung gegen die Quelloffenheit seines Systems getroffen. Bei Johans Vertretungsplan-System gibt es nur einen zentralen Server, auf den alle Schüler über die Android-App zugreifen. Je mehr Schulen seine App nutzen, umso teurer werden auch die Unterhaltungskosten des Servers. Da diese aber kostenlos erhältlich war und er somit keine Einnahmen hatte, musste eine Lösung gefunden werden. Er entschied sich dafür, einzelne Funktionen der App durch In-App-Käufe kostenpflichtig zu machen und deshalb den Quellcode nicht mehr zu veröffentlichen, weil dadurch die Kostenhürde umgangen werden könnte.

Zusätzlich zu den In-App-Käufen bietet Johan eine Schullizenz an, die es Schulen ermöglicht, Premiumfunktionen für alle Schüler freizuschalten und die Weiterentwicklung der Software zu unterstützen. Durch die Einnahmen, die Johan jetzt schon erzielt hat, war er in der Lage, die Entwicklung einer iOS-App zu finanzieren.

Johan bot mir aber eine Alternative an: Er lagerte die Teile des Servers, die die Daten der Vertretungspläne einholt und in ein einheitliches Format umwandelt, in eine Bibliothek aus, die er anschließend unter einer freien Lizenz auf GitHub veröffentlichte: Substitution-Parser-Library [32].

Nachdem Johan den Quellcode veröffentlicht hatte, fing ich sofort mit der Erstellung eines Parsers für LegionBoard an [33] und erhielt dabei viel Unterstützung von Johan. Nach kurzer Zeit war die Arbeit beendet und so kann man seit dem 4. August 2016 das LOG als Schule in der App auswählen [34].

Durch die Android-App unterstützt LegionBoard damit indirekt eine Bedingung, die ich zuvor als wünschenswert für Vertretungsplansysteme aufgestellt habe: Benachrichtigungen. Johans Server holt in regelmäßigen Abständen die Änderungen von LegionBoard ab und informiert die Schüler über Firebase Cloud Messaging (FCM) [35] bei Änderungen am Vertretungsplan.

An dieser Stelle möchte ich Johan für die tolle Zusammenarbeit danken!

6 Zukunft

Nach langer Zeit ist die Entwicklung von LegionBoard nun abgeschlossen. Nunja, zumindest die Entwicklung von Version 0.2.0. Zu dem Zeitpunkt, an dem ich diese Zeilen verfasse, ist die Version 0.2.0 die letzte Veröffentlichung von Heart und Version 0.2.3 die letzte von Eye. Eigentlich hatte ich gedacht, dass Version 0.1.0 für längere Zeit die aktuelle Veröffentlichung von LegionBoard sein wird und hatte zu diesem Zweck extra mit der Veröffentlichung gewartet. Erst nach zwei Wochen Einsatz am LOG veröffentlichte ich Version 0.1.0 und damit den Quellcode von LegionBoard, doch auch nach dieser Zeit hielt die Weiterentwicklung nicht an und so wurde bald danach Version 0.2.0 veröffentlicht.

In letzter Zeit gab es keine großen Änderungen an LegionBoard, was zum Teil daran liegt, dass das Abitur immer näher rückt, aber auch daran, dass die Software jetzt aus den Kinderschuhen heraus ist und einfach läuft. Es existieren zwar einige Punkte, die verbessert werden könnten [36], doch der Betrieb im Schulalltag ist nicht eingeschränkt.

Wie schon im letzten Kaptiel besprochen, ist der Quellcode von LegionBoard komplett einsehbar. Außerdem wird es hundertprozentig von Johans Substitution-Parser-Library unterstützt und funktioniert damit auch unter Android und iOS reibungslos. Dazu kommt die Unterstützung von LANiS, die ich im Unterabschnitt 6.4 beschreibe.

Wegen all diesen Punkten hoffe ich, dass in Zukunft mehr Leute zu LegionBoard beitragen und es ständig weiterentwickelt wird.

6.1 Weiterentwicklung seit Version 0.2.0

Bis Version 0.2.0 habe ausschließlich ich an LegionBoard gearbeitet, doch in letzter Zeit gab es auch einen Beitrag meines Freundes Jan Weber. Zusammen mit mir hat er daran gearbeitet, dass Heart nun auch Fächer unterstützt. In zukünftigen Versionen wird es also möglich sein, bei einer Änderung ein Fach zu definieren und auch Kurse und Lehrer mit Fächern zu verknüpfen, sodass diese dann direkt vorgeschlagen werden. Außerdem unterstützt Heart nun einen neuen Änderungstypen: Raumwechsel.

Bisher wurde jedoch noch keine neue Version mit diesen Änderungen veröffentlicht.

In Eye gab es mit Version 0.2.1 zwei Änderungen: So werden jetzt die beim Filtern ausgewählten Kurse und Lehrer gespeichert, sodass man diese nur einmal auswählen muss und dann nur noch relevantere Informationen angezeigt bekommt. Dazu kommt, dass das exakte Datum versteckt wird und nur der Wochentag angezeigt wird, wenn sich die Änderung in der aktuellen Woche befindet. Statt „Mo, 20.2.17„ wird also nur „Montag“ angezeigt.

Diese beiden Änderungen verbessern die Benutzungserfahrung sehr.

Nachdem es in Version 0.2.2 nur kleinere Verbesserungen gab, wurde in Version 0.2.3 ein lange existierendes Problem gelöst: Wie zuvor erwähnt, gab es nicht-reproduzierbare Fälle auf iOS- und macOS-Geräten, die dazu führten, dass die Zugangsdaten als falsch abgewiesen wurden, obwohl sie stimmten. Als ich in letzter Zeit endlich die Möglichkeit bekam, ein Gerät mit diesem Fall zu debuggen, also nach dem Fehler zu suchen, fand ich die Ursache. In Safari gibt es einen „privaten Modus“, der dazu führt, dass die Speichermethode, die Eye bevorzugt verwendet (HTML5 Web Storage [16]), nicht funktioniert und Eye dann nicht wie erwartet auf Cookies umstellt, sondern einfach den Dienst verweigert. Der Algorithmus zum Fallback wurde nun verbessert und seitdem kam es zu keinen erneuten Problemen mit iOS/macOS.

Für jede Veröffentlichung von Heart und Eye exisitert ein Eintrag im Changelog, die an folgenden Stellen zu finden sind:

- gitlab.com/legionboard/heart/blob/master/CHANGELOG.md
- gitlab.com/legionboard/eye/blob/master/CHANGELOG.md

6.2 App für mobile Geräte

Dank Johans Substitution-Parser-Library unterstützt LegionBoard indirekt auch Android und iOS nativ, doch dessen Apps sind abhängig von seinem zentralen Server und zudem aufgrund der In-App-Käufe nicht Open Source.

Insofern wäre es wünschenswert, wenn es eine eigene App für LegionBoard gäbe, die die Daten direkt von Heart abholt und, wie alle anderen Teile von LegionBoard, vollständig Open Source wäre.

Da frühere Versionen der Android-App von Vertretungsplan.me quelloffen sind, bietet sich dies als Basis für die neue LegionBoard-App an. Ich habe bereits einen Fork angelegt, der den letzten öffentlichen Code der Android-App enthält, und ihn auf GitLab hochgeladen: gitlab.com/legionboard/android

Bisher habe ich jedoch an der alten App nichts geändert, sodass sie im Moment nutzlos ist.

Eine Version für iOS wäre sicherlich auch wünschenswert, doch die Entwicklungskosten dafür sind hoch. Während ein Entwickler Android-Apps kostenlos mit Android Studio auf Linux, Windows und macOS programmieren kann und diese dann gegen einen einmaligen Pfand bei der Registrierung auf Google Play hochladen kann, wird zur Entwicklung einer iOS-App erstmal ein Mac benötigt. Hat man ein Gerät mit macOS, kann man zwar auch kostenfrei Apps mit xCode entwicklen, doch für die Veröffentlichung im App Store bittet Apple die Enwickler mit jährlich 100 US-Dollar zur Kasse.

Aus diesem Grund ist eine LegionBoard-App für iOS in nächster Zeit unwahrscheinlich.

6.3 Bandbreite sparen durch Hashvergleich

Im Moment lädt Eye bei jeder Aktualsierung die kompletten Daten von Eye herunter. Man könnte Bandbreite sparen, indem Eye nicht sofort die Änderungen herunterlädt, sondern erst einmal Heart fragt, ob sich etwas geändert hat. Dazu schickt Heart bei jeder Anfrage einen Hash mit, der die aktuelle Version des Vertretungsplans darstellt und von Eye gespeichert wird.

Wenn Eye dann die aktuellen Daten abfragt, schickt es diesen Hash bei der Anfrage mit und bekommt dann wahlweise die aktuellen Daten oder einen Hinweis, dass sich seit dem letzten Mal nichts verändert hat.

Die Entwicklung dieser Funktion kann man in folgendem Issue auf GitLab verfolgen: gitlab.com/legionboard/heart/issues/24

6.4 LANiS

Schulen in Hessen haben kostenfrei die Möglichkeit, die Administration ihrer Netzwerke mit LANiS (Leichte Administration von Netzwerken in Schulen) durchzuführen. Ein Teil davon ist die Plattform „LANiS-Online“ [37], die

eine Vielzahl von hilfreichen Werkzeugen in einer einheitlichen Oberfläche vereint.

Eines dieser Werkzeuge ist der „Personalisierte Vertretungsplan“. Dabei wird dem Nutzer nicht der gesamte Vertretungsplan mit allen Kursen und Lehrern angezeigt, sondern nur die Informationen, die ihn betreffen. LANiS-Online unterstützt für diese Funktion mehrere verschiedene Systeme und dank meines Informatik-Lehrers Herrn Richter hatte ich die Möglichkeit, mit dem leitenden Entwickler von LANiS über LegionBoard zu sprechen.

Dieser zeigte sich sehr offen gegenüber einer Unterstützung von LANiS-Online und stellte nur eine Bedingung: alle relevanten Daten (Kurse, Lehrer, Änderungen einer Woche) sollten mithilfe einer einzigen Anfrage an Heart abrufbar sein. Ich erklärte mich bereit, diese Funktion nach dem Abitur zu implementieren und hoffe, dass danach schnell eine Unterstützung seitens LANiS implementiert werden kann, da dies das Projekt mit einem Schlag um einiges bekannter machen würde und dann sicherlich andere Schulen auf LegionBoard aufmerksam werden würden.

Der aktuelle Status der Funktion zum Gesamtexport lässt sich in folgendem Issue verfolgen: gitlab.com/legionboard/heart/issues/25

7 Abschlusskommentar

LegionBoard erfüllt jetzt alle Anforderungen, die ich weiter oben an ein modernes Vertretungsplansystem gestellt habe: es ist für mobile Geräte optimiert, da sich Eye an kleine Bildschirme anpasst und wenig Daten pro Aufruf übertragen werden; es ist sicher, da bei jedem Aufruf die Rechte des Benutzers geprüft werden und der Datenbankzugriff von der visuellen Darstellung getrennt ist; durch die Aufteilung in Ressourcen sind selbst komplexe Datenstrukturen einfach verwaltbar; einzig die Benachrichtigung bei Neuerungen in den Änderungen ist bisher noch nicht nicht nativ in LegionBoard umgesetzt. Diesen Punkt und wie die App von Vertretungsplan.me als dessen zwischenzeitliche Lösung agiert habe ich im Unterabschnitt 5.1 erläutert.

Literatur

[1] *REST API*, um auf Google Fit zuzugreifen.

developers.google.com/fit/rest

[2] *REST API*s, um auf Twitter zuzugreifen.

dev.twitter.com/rest/public

[3] *REST API Graph*, um auf Inhalte in Facebook zuzugreifen.

developers.facebook.com/docs/graph-api

[4] Roy Thomas Fielding *Architectural Styles and the Design of Network-based Software Architectures* 2000

[5] LegionBoard Heart benutzt mehrere Apache-Konfigurationen, um künstliche Weiterleitungen zu bauen und Zugriff auf interne Strukturen zu verhindern.

gitlab.com/legionboard/heart/blob/master/src/.htaccess

gitlab.com/legionboard/heart/blob/master/src/lib/.htaccess

[6] Die Darstellung des Vertretungsplans darf nicht öffentlich erfolgen, sondern unterliegt verschiedenen Bestimmungen: „Der datenschutzrechtliche Ausgangspunkt ist § 83 Abs. 1 HSchG. [...] Der Vertretungsplan kann ggf. zusätzlich auf einer Plattform einer geschlossenen Benutzergruppe (Schüler, Eltern und Lehrer) aber keinesfalls öffentlich im Internet zur Verfügung gestellt werden.“

Quelle: datenschutz.hessen.de/ds009.htm

[7] Die Erstellung von Benutzern/Authentifizierungsschlüsseln erfordert das manuelle Hochladen eines *PHP*-Skripts auf den Server.

gitlab.com/legionboard/heart/blob/master/src/lib/tools/createUser.php

[8] Skeleton ist ein leichtgewichtiges Framework, mit dem man einige hilfreiche Werkzeuge in unter 400 Zeilen Code bekommt.

getskeleton.com

[9] Meine von GitHub [28] gehostete Webseite, auf der ich einzelne Projekte vorstelle, an denen ich mitgewirkt habe.

altnico.github.io

[10] Bootstrap ist ein sehr populäres Framework, dass von Millionen Webseiten eingesetzt wird und unzählige Funktionen bietet. Im Grunde ist es, zusammen mit einigen Erweiterungen, die eierlegende Wollmilchsau unter den Frameworks.

getbootstrap.com

[11] Das „Dropdown Menus Enhancement“ für Bootstrap erweitert die Funktionen der Menüs, die in Eye dafür benutzt werden, Lehrer und Kurse auszuwählen. Konkret wird hier die Möglichkeit genutzt, sogenannte „radio and checkboxes“ in Menüs zu verwenden.

github.com/behigh/bootstrap_dropdowns_enhancement

[12] Um das Datum in Eye nicht manuell eingeben zu müssen, wird in Eye die Bootstrap-Erweiterung „Datepicker“ benutzt. Damit öffnet sich bei einem Klick auf das Datums-Feld ein Fenster, in dem man das Datum aus einem Kalender auswählen kann.

github.com/uxsolutions/bootstrap-datepicker

[13] Jede Funktion, die Eye benutzt, in JavaScript zu schreiben, wäre ein sehr großer Aufwand, da man dabei darauf achten muss, dass der Code auch von jedem Browser unterstützt wird.

Um die Entwicklung von Eye zu vereinfachen, habe ich mich deshalb dafür entschieden, jQuery zu verwenden.

jquery.com

[14] Die standardmäßig in Browsern enthaltenen Dialog-Fenster sind nicht sehr schön. Um bei Erfolgs- und Fehlermeldungen eine bessere Benutzungserfahrung anzubieten, benutzt Eye die Bibliothek „SweetAlert“.

github.com/t4t5/sweetalert

[15] LegionBoard Eye teilt dem Server mit, dass einzelne Dateitypen vom Gerät des Benutzers zwischengespeichert werden können.

gitlab.com/legionboard/eye/blob/master/src/.htaccess

[16] Der *Web Storage* hat mit HTML 5 Einzug in die Browser gehalten. Ähnlich wie bei Cookies kann man hiermit Daten auf dem Gerät des Benutzers abspeichern, jedoch werden diese nicht an den Server geschickt, sondern sind nur über JavaScript lokal verfügbar.

w3schools.com/HTML/html5_webstorage.asp

[17] Es ist geplant, in Zukunft die Berechtigungen eines Nutzers abzufragen und ausschließlich Aktionen anzuzeigen, zu denen er berechtigt ist.

gitlab.com/legionboard/eye/issues/26

[18] „Der Quelltext wurde veröffentlicht, damit jede Schule rund um den Globus es frei benutzen, studieren und modifizieren kann.“

legionboard.org/index_de.html#overview

[19] Auf der Webseite befinden sich Links zu Installationsanleitungen von LegionBoard.

legionboard.org/index_de.html#Install

[20] Auf der offiziellen Webseite von LegionBoard befindet sich die aktuellste Version von KISS und man kann damit den Vertretungsplan jeder beliebigen Schule aufrufen.

legionboard.org/kiss

[21] Quelloffene und freie Android App zum Verwalten von Hausaufgaben.

gitlab.com/hw-manager/android

[22] HA-Manager ist heute in elf Sprachen übersetzt.

crowdin.com/project/hw-manager

[23] Die offizielle Webseite von LegionBoard informiert über das System und enthält unter anderem Screenshots und Links zu Installationsanleitungen.

legionboard.org

[24] Diese Lizenz wurde am MIT entwickelt und wird von mir vor allem für kleinere Projekte verwendet.

opensource.org/licenses/MIT

[25] Die „General Public License“, kurz „GPL“, ist die Standardlizenz für Software der GNU-Reihe und wird von mir für alle umfangreicheren Projekten verwendet, von denen ich möchte, dass der Quellcode von Modifikationen ebenfalls veröffentlicht wird.

gnu.org/licenses/gpl.html

[26] Die Free Software Foundation ist eine gemeinnützige Organisation, die sich für freie Software einsetzt. Sie wurde von Richard Stallman gegründet, der als Pionier der freien Software gilt.

fsf.org

[27] Git ist ein dezentrales Versionsverwaltungssystem, das seine Vorgänger schnell überholt hat und heute von so gut wie jedem Entwickler eingesetzt wird.

git-scm.com

[28] GitHub ist die zur Zeit populärste Webseite zum Hosten von Quellcode. Während sich hier so gut wie jede freie Software finden lässt, ist die unterliegende Webseite leider nicht quelloffen. Aus diesem Grund wird LegionBoard auf GitHub nur gespiegelt und stattdessen auf GitLab [29] gehostet.

github.com

[29] GitLab ist ein Newcomer in der Code-Hosting-Szene und hat von seiner Veröffentlichung im Jahr 2012 bis heute einen erstaunlichen Start hingelegt. Im Gegensatz zu GitHub ist die Community Edition vollständig quelloffen und wird deshalb von immer mehr freien Projekten bevorzugt benutzt.

gitlab.com

[30] Die englischsprachige Dokumentation der API von LegionBoard Heart.

gitlab.com/legionboard/heart/blob/master/doc/README.md

[31] Der Physik-Student Johan von Forstner hat eine Software geschrieben, die die Vertretungspläne anderer Systeme importiert und dann einheitlich in einer Android-App darstellt.

Leider sind neuere Versionen der Software nicht mehr Open Source, da die Finanzierung des Servers von den In-App-Käufen der Android-App abhängt.

Android-App: github.com/johan12345/vertretungsplan

Server: github.com/johan12345/vertretungsplan-server

[32] Substitution-Parser-Library ist eine Bibliothek von Johan von Forstner, mit der man Vertretungspläne aus verschiedenen Systemen importieren und einheitlich darstellen kann. Sie ist unter der Mozilla Public License Version 2.0 auf GitHub veröffentlicht:

github.com/johan12345/substitution-schedule-parser

[33] Die Bibliothek von Johan Forstner unterstützt auch LegionBoard.

github.com/johan12345/substitution-schedule-parser/commits/master/parser/src/main/java/me/vertretungsplan/parser/LegionBoardParser.java

[34] Seit dem 04. August 2016 ist das LOG als Schule in der Vertretungsplan-App von Johan von Forstner als Schule auswählbar.

vertretungsplan.me/2016/08/26/legionboard-indiware.html

[35] Firebase Cloud Messaging (FCM) ist der Nachfolger von Google Cloud Messaging (GCM) und wird unter anderem von Johans Vertretungsplan-App dazu benutzt, Schüler über Änderungen am Vertretungsplan zu informieren.

firebase.google.com/docs/cloud-messaging

[36] Auch nach der Veröffentlichung von Version 0.2.0 gibt es einige Punkte, die an LegionBoard verbessert werden können.

gitlab.com/groups/legionboard/issues

[37] LANiS-Online ist „die Plattform für die pädagogischen und organisatorischen Online-Angebote einer Schule“ und für Schulen in Hessen kostenfrei nutzbar.

lanis-system.de/ueber-lanis/uber-lanis

lanis-system.de/ueber-lanis/lanis-online