WahlsystemBiologie Dokumentation

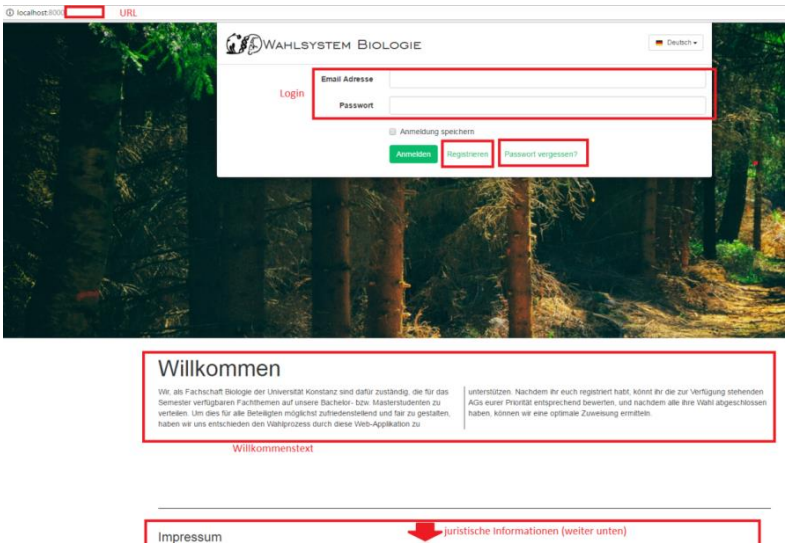
Installation

Nach dem klonen des Git Repositories, muss composer update ausgeführt werden. Weitere Maßnahmen müssen nicht getroffen werden, um die Applikation vorzubereiten.

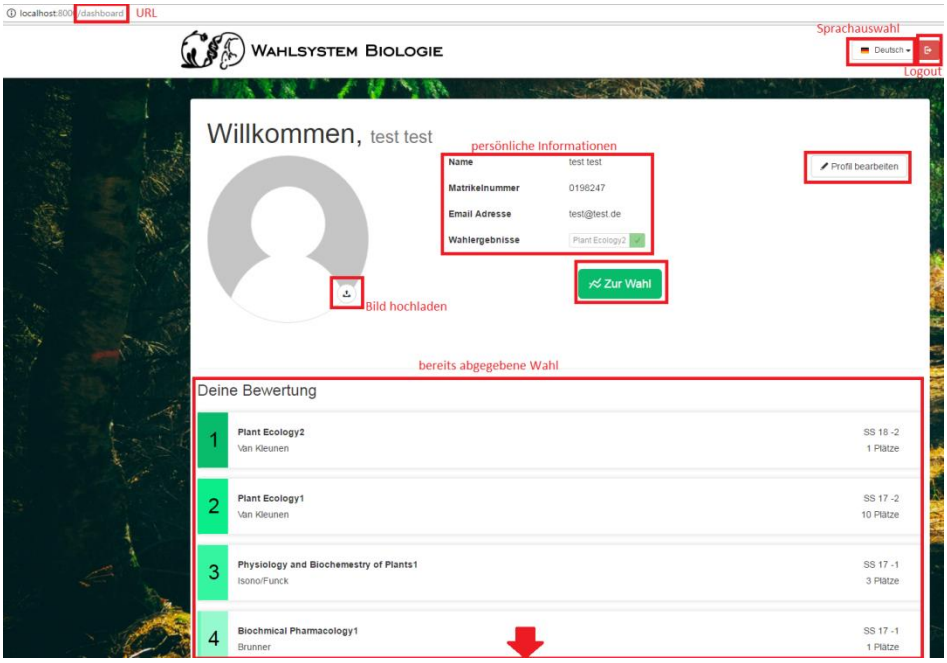
Übersicht und Einführung in die Applikation

Schon seit einiger Zeit wird der Biologie Fachschaft der Universität Konstanz die Aufgabe übertragen, unter den Studierenden, welche ihre Bachelor- bzw. Masterarbeit schreiben, die zur Verfügung stehenden Themen(Arbeitsgruppen) zu verteilen. Dabei sollen nicht einfach Themenwillkürlich an die Studierenden verteilt werden, sondern die Studierenden sollen die zur Verfügung stehenden Themen nach ihrer Präferenz bewerten können. Aus diesen Bewertungen soll dann eine möglichst zufriedenstellende Verteilung errechnet werden. Das Ziel der Anwendung ist es, dass jeder Student ein Thema, mit möglichst hoher Bewertung erhält. Um dies zu erreichen, benötigt die Fachschaft ein Programm, welches wir als Web-Applikation realisiert haben. Diese Applikation ermöglicht es den Studierenden, ihre persönlichen Bewertungen abzugeben, diese algorithmisch auszuwerten und die optimale Zuteilung der Studierenden auf die Themen auszugeben. Dabei war das Haupt-Qualitätsmerkmal der Applikation die Usability und Stabilität, die Fachschaft hatte sich eine komfortable GUI zum Verwalten der Wahl gewünscht.

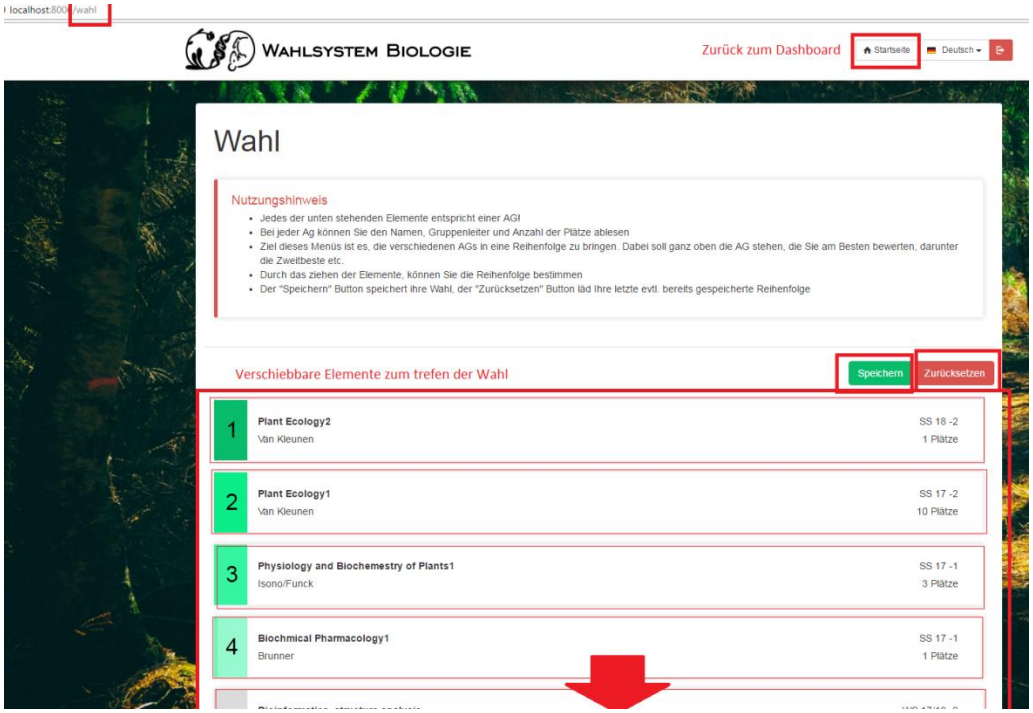
Die Applikation ist in zwei Bereiche eingeteilt, welche getrennt voneinander zu erreichen sind. Der eine Teil ist der Userbereich, welche für jeden angemeldeten Studierenden einzusehen ist. Der zweite Bereich ist nur für die vorher definierten Administratoren einzusehen. Wenn man sich mit den Daten eines Administrators auf der Landingpage einloggt, wird man dementsprechend weitergeleitet. Auf der Landingpage wird dem Nutzer ein Willkommenstext angezeigt, dieser kann im Admin Menü dynamisch verändert werden. Weiterhin stehen juristische Informationen sowie das Formular für Registrierung und Login zur Verfügung. Zusätzlich besteht die Möglichkeit sein eigenes Passwort per Mail zurücksetzen zu lassen.



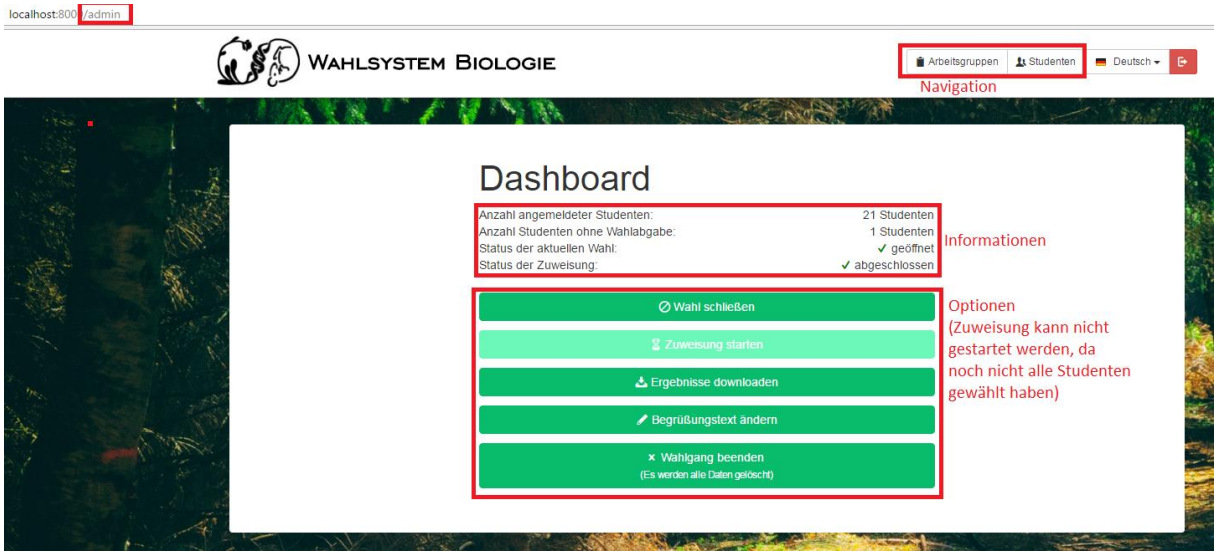
Im Userbereich werden dem angemeldeten Studierenden seine persönlichen Daten angezeigt, wobei er die Möglichkeit hat diese, solange die Wahl noch geöffnet ist, zu bearbeiten. Nachdem der Student die Arbeitsgruppen nach seiner Präferenz geordnet hat, wird ihm diese auch angezeigt. Zusätzlich besteht die Möglichkeit ein Profilbild hochzuladen, welches persistent gespeichert wird. Da die Studenten, vor allem im Master, zu einem großen Teil aus dem Ausland stammen, haben wir die Applikation bilingual eingerichtet, wobei diese Funktion auch im Admin Bereich zur Verfügung steht.



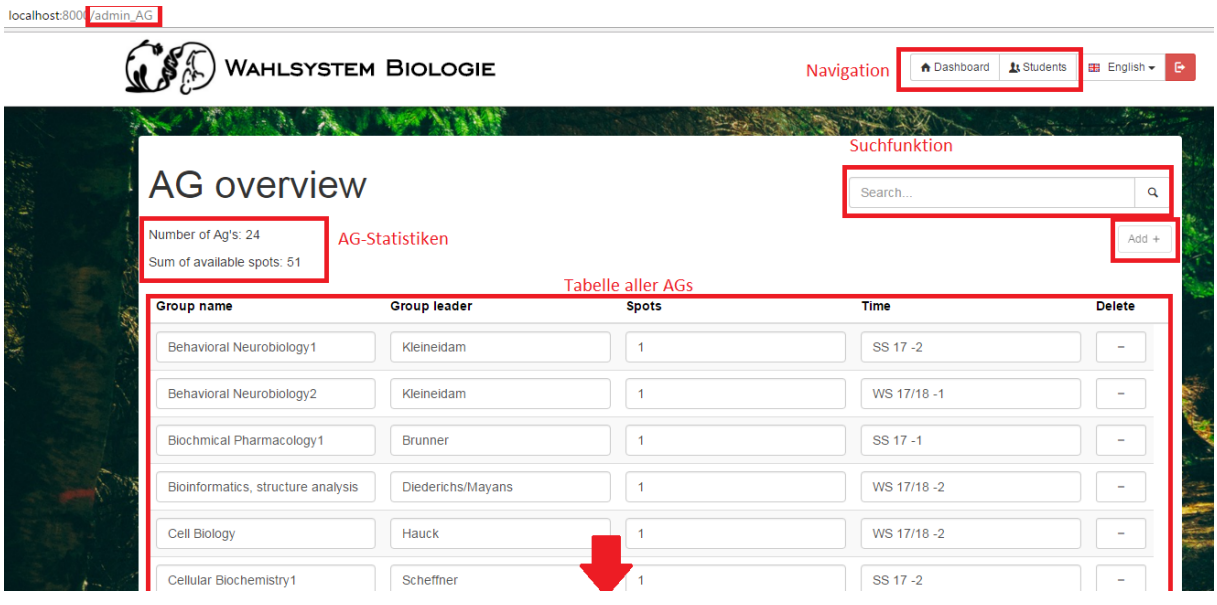
Die Hauptfunktion des Userbereichs liegt darin, dass eine Wahl getätigt werden kann. Die entsprechende Weiterleitung ist auf dem User Dashboard verfügbar. Das Menü zum Wählen besteht aus einer Anzahl von Elementen, welche durch Drag&Drop in eine Reihenfolge gebracht werden sollen, dabei ist das oberste Element die am höchsten bewertete Arbeitsgruppe.



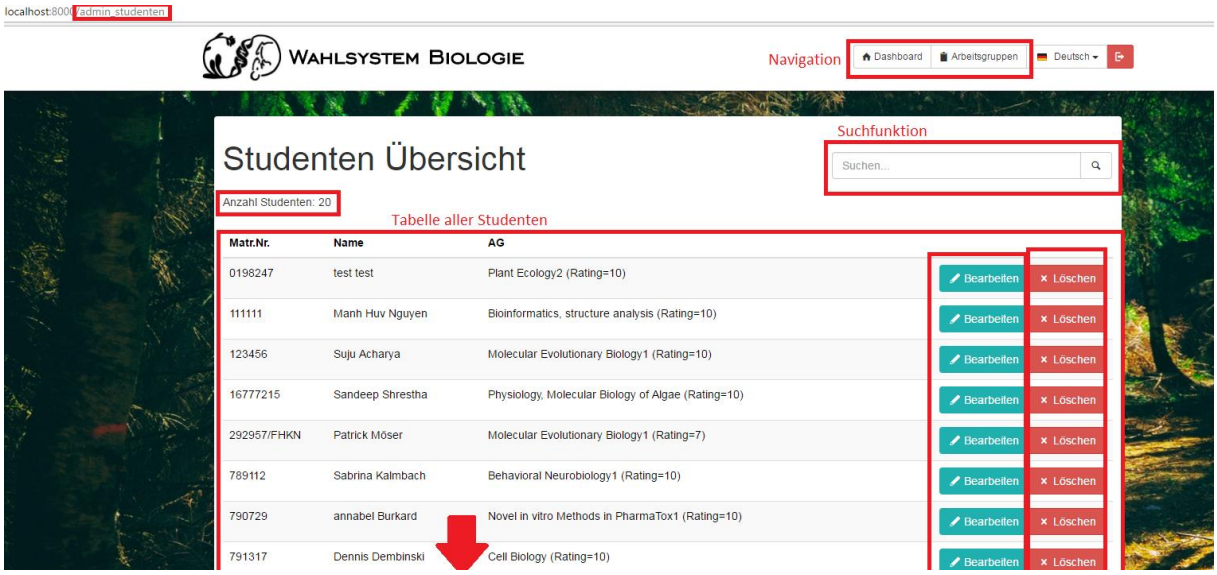
Nach dem Login des Administrators (Email: mat.leo@web.de, Passwort: 88888888) wird man auf das Admin Dashboard geführt, h. Hier stehen die wichtigsten Funktionalitäten der Applikation zur Verfügung. Je nach Fortschritt des Wahlvorgangs werden dem Administrator Informationen angezeigt, so kann er die Wahl schließen oder wieder öffnen, die Zuweisung starten, die Ergebnisse als Excel Sheet downloaden oder bequem Daten aus der Datenbank löschen. Zudem besteht hier die Möglichkeit den Willkommenstext der Landingpage zu bearbeiten. Über die Navigation am oberen Rand können die beiden Übersichtsseiten aufgerufen werden.



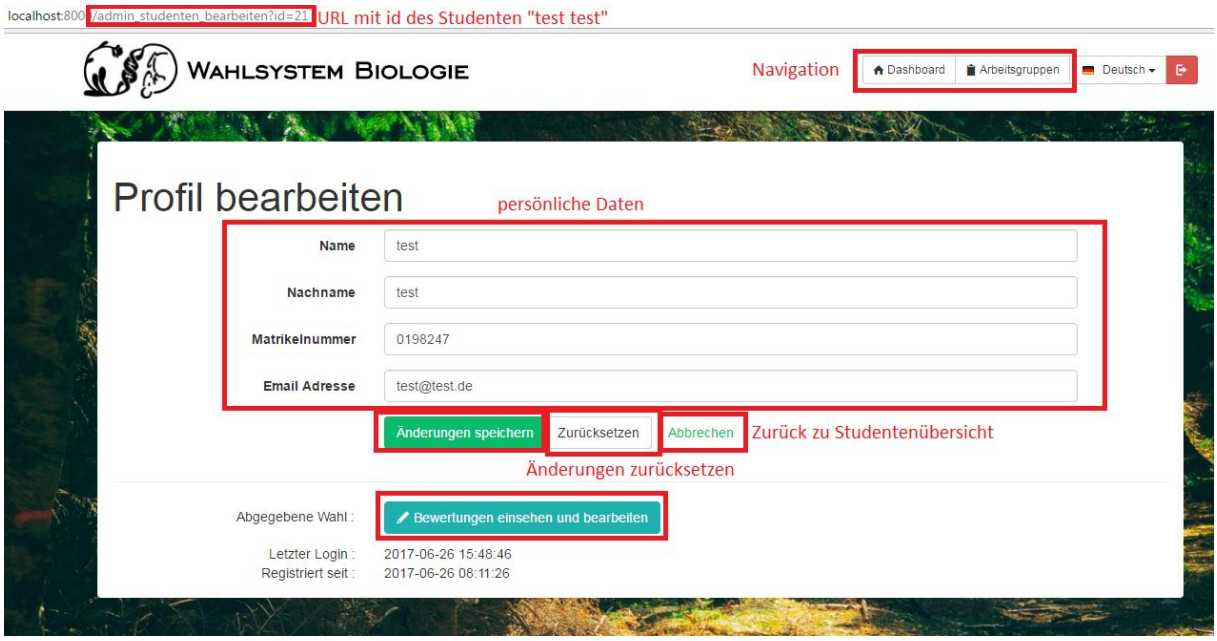
Zum einen kann eine Übersicht der eingetragenen Arbeitsgruppen aufgerufen werden. Dort können neue Arbeitsgruppen hinzugefügt oder bestehende bearbeitet bzw. gelöscht werden. Sobald der erste Student eine Wahl abgegeben hat, können keine neuen AGs hinzugefügt werden, da ansonsten die bereits getätigte Wahl unvollständig wäre. Soll eine AG gelöscht werden für die bereits Bewertungen existieren, werden diese ebenfalls gelöscht. Dabei werden nicht alle Bewertungen der betroffenen Studierenden gelöscht, sondern nur die Datensätze der gelöschten Arbeitsgruppe. Zusätzlich können Statistiken über die Anzahl der Arbeitsgruppen und die gesamte Anzahl an verfügbaren Plätzen entnommen werden. Des Weiteren können mit Hilfe der Suchfunktion alle Arbeitsgruppen nach Name, Leiter, Plätze oder Zeitpunkt aussortiert werden.



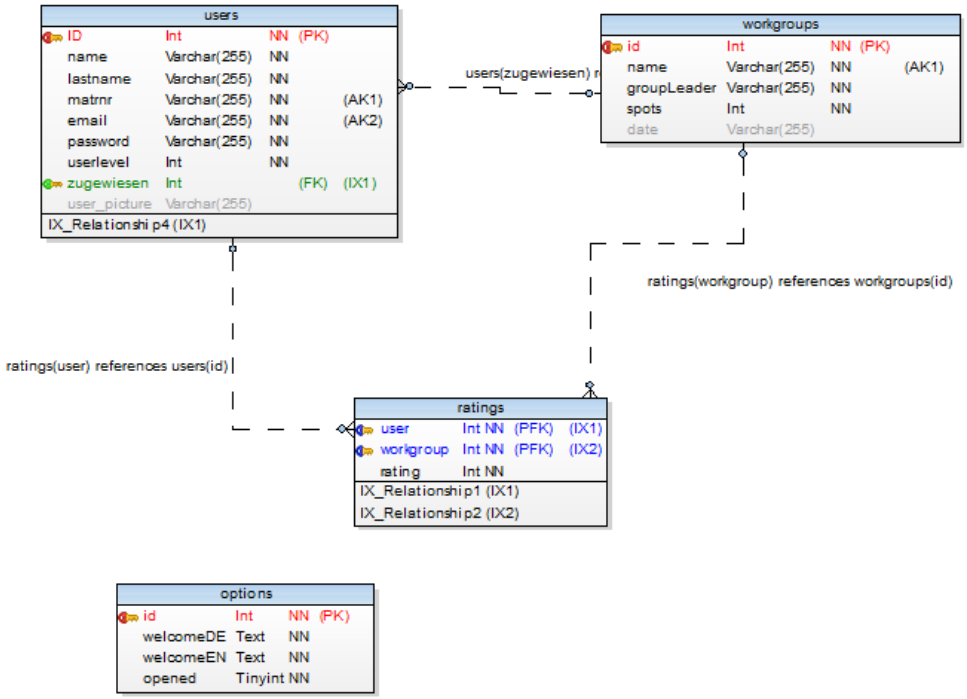
In der Studentenübersicht werden alle registrierten Studenten aufgelistet. Zudem besteht die Möglichkeit die persönlichen Daten sowie die Wahl der einzelnen Studenten einzusehen sowie Studenten zu löschen. Hierbei gibt es wieder eine Suchfunktion, die es ermöglicht Studenten nach Matrikelnummer, Vorname, Nachname oder zugewiesener Arbeitsgruppe zu durchsuchen.



Mit einem Klick auf einen der „Bearbeiten“-Buttons, werden die Daten des Studierenden angezeigt. Diese können verändert werden, allerdings dürfen die Matrikelnummer und Email Adresse hierfür nicht bereits in der Datenbank existieren. Zudem können in diesem Bereich auch alle abgegebenen Bewertungen eingesehen und verändert werden.



Datenbank-Struktur



Neben „*migrations*“ und „*password_resets*“, welche von Laravel automatisch angelegt werden, wurden vier weitere Datenbanktabellen angelegt. Zusätzlich zu den dargestellten Attributen besitzt jede Tabelle noch die Attribute *created_at* und *updated_at*. Der Anwender verfügt für die Wiedererkennung einen *remember_token*.

In der Tabelle „*users*“ werden alle registrierten Anwender sowie alle Administratoren gespeichert, Admin User müssen dabei direkt per SQL erzeugt werden. Nutzerkategorien werden anhand des Attributs „*userlevel*“ unterschieden, wobei *userlevel=0* für einen normalen Anwender und *userlevel=1* für einen Administrator steht. „*Zugewiesen*“ hat einen Fremdschlüssel auf *workgroups* und speichert die zugewiesene Arbeitsgruppe eines Studenten, nach der Zuweisung. Im Attribut „*user_picture*“ wird eine Referenz auf das hochgeladene Bild des Users gespeichert.

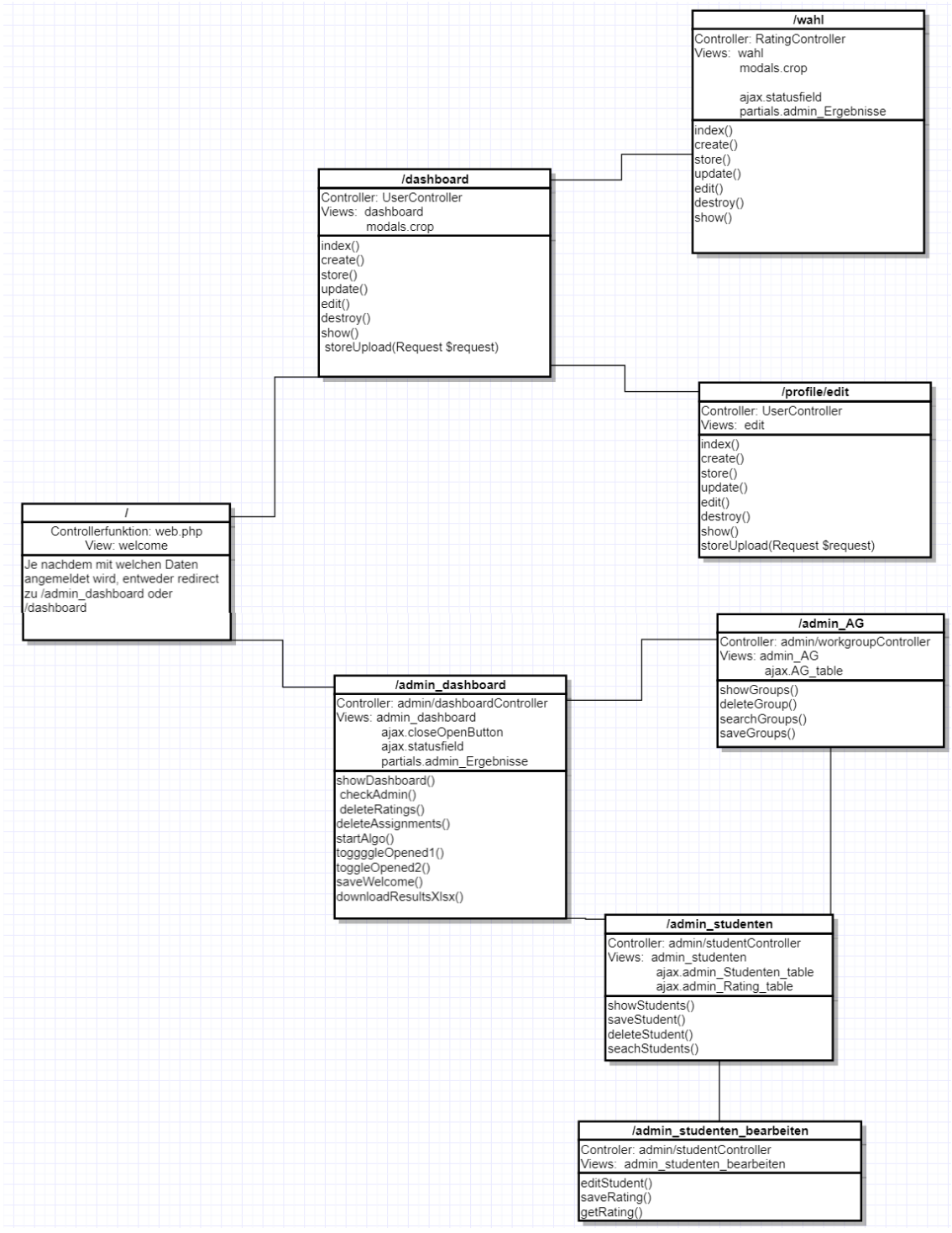
In der Tabelle „*workgroups*“ werden alle von der Fachschaft eingetragenen Arbeitsgruppen gespeichert. Im Attribut „*date*“ wird bei einer Masterwahl das Semester bzw. Datum gespeichert, indem diese AG angeboten wird.

In der Tabelle „*ratings*“ werden die Bewertungen der Studenten gespeichert. Dabei muss jeder Student alle Arbeitsgruppen bewertet haben, bevor eine Wahl persistiert wird. Das Attribut ist ein Foreign Key auf „*users(id)*“, das Attribut „*workgroup*“ auf „*workgroups(id)*“. “*Rating*” ist die Zahl, wie hoch ein Student eine AG bewertet hat, wobei 10 der höchsten und 1 der niedrigsten Wertung entspricht.

Die Dummy-Tabelle „*options*“ besitzt nur eine Zeile. Das Attribut „*welcomeDE*“ ist dabei der deutsche Willkommenstext, welcher auf der Landingpage angezeigt wird und „*welcomeEN*“ der Englische. Das Attribut „*opened*“ nimmt entweder den Wert 0 an, falls die Wahl geschlossen ist, oder 1, falls die Wahl geöffnet ist. In der Laravel Migration hatten wir das Attribut als boolean definiert, dieser wurde in MySQL in tinyInt umgewandelt.

Applikations-Struktur

Im Folgenden ist eine Abbildung zu sehen, welche die Seitenstruktur des Admin Bereichs und die des User Bereichs darstellt, dabei wird je nach eingegebenen Login Daten auf der Landingpage, entweder auf die Admin oder auf die User Seite weitergeleitet. Die Struktur der Diagramme lehnt sich an UML-Klassendiagramme an. Die Überschrift entspricht der URL, durch welche die Seite aufgerufen wird. In dem darunterliegenden ersten Abschnitt ist der zugehörige Controller und die genutzten Views aufgelistet. Der letzte Abschnitt ist mit den sprechenden Methodennamen aus den Controllern gefüllt, um einen ersten Eindruck zu geben, welche Funktionalität auf welcher Seite implementiert ist.



Beschreibung des Designs

Struktur

Bei der Gestaltung der Struktur werden aktuelle und moderne Elemente eingebaut. Neben einem direkten Login Bildschirm, der sich über die fast gesamte Browserhöhe erstreckt, finden sich kleine informative Sektionen. Der Login Bildschirm wird bewusst auf diese Größe gesetzt um dem Anwender die Kernfunktionalität der Web-Applikation zu zeigen. Er muss sich, um diese nutzen zu können, Authentifizieren. In den informativen Sektionen stehen Informationen zu aktuellen Wahlergebnissen oder ob die Wahl von der Fachschaft bereits ausgetragen wurde.

Nach der Authentifizierung erhält ein Benutzer die Möglichkeit sich über die Navigation in der Applikation zurecht zu finden. Er erhält zudem in einer Übersicht Informationen zu seinen privaten Daten, Wahlergebnis und Rangfolge seiner Wahl.

Die Struktur der Anwendung wird durch die beiden Blade-Templates: *app* und *admin* aufrechterhalten. Beide Templates sind unter *resources/view/layouts* zu finden. Sie bieten beide die Möglichkeit für Seitenspezifisches CSS und JavaScript.

Aufgrund der Teilung von Struktur und Seitenspezifischen Inhalten, können wir uns zu dem eine dynamische Navigation zusammenbauen. So erhält ein Anwender, der auf seinem Dashboard ist, einen Link zur Wahl, Sprachauswahl und um sich abzumelden. Während ein Anwender innerhalb der Wahl, anstelle des Verweises zur Wahl, einen Verweis zum Dashboard findet.

Designentwicklung

Bei der Entwicklung des Designs wird versucht der Fachschaft Biologie eine neue Anwendung zu liefern, die jedoch in der Oberflächengestaltung bekannte Elemente verwendet.

So kann das Logo der Fachschaft auch in unserer neuen Anwendung auf jeder Seite als Logo gefunden werden. Um einen Kontrast zu den bisher hellen Farben zu erreichen, wurde ein dunkles Hintergrundbild verwendet. Auch hier erkennt man den Bezug zur Natur. Die bisherige Anwendung verwendete Grüntöne, welche ebenfalls aufgegriffen werden.

Aufgrund des dunklen Hintergrundbildes wird der Inhalt der Seite durch helle Containerelemente zentriert hervorgehoben.

Implementierung der Applikation

Admin Bereich

Im Folgenden wird die Implementierung auf einem hohen Abstraktionslevel beschreiben. Im Detail ist die programmiertechnische Implementierung den Kommentaren im Quellcode zu entnehmen.

Admin Dashboard

Das Dashboard ist durch die URL „*/admin_dashboard*“ aufzurufen. Der Controller *admin/dashboardController* ruft initial mit der Methode *showDashboard()* die View *admin_dashboard* auf und übergibt die Parameter, welche auf der Seite dargestellt werden.

Zum Beispiel wird ein Modal aufgerufen, wenn die Seite nach dem Starten des Algorithmus oder Beenden der Wahl neu geladen wird. Erst wenn alle registrierten Studenten ihre Wahl abgegeben haben, kann die Zuweisung gestartet werden. Die Anzahl der Studenten ohne Wahlabgabe wird angezeigt, sollte sie *>0* sein und als Differenz zwischen der Gesamtanzahl Studenten und den *distinct usern* aus der tabelle *ratings* berechnet. Falls es Studierende ohne Wahlabgabe gibt, werden diese bei einem Klick auf den *disabled* Zuweisen-Button angezeigt. Diese Studierenden lassen sich mit Hilfe eines *outer joins* zwischen *users* und *ratings* berechnen, indem das Attribut *ratings.user* auf *NULL* überprüft wird.

Bei dem Betätigen des „*Wahlgang beenden/öffnen*“ Buttons, wird die JavaScript Funktion *toggle()* aufgerufen, welche jeweils AJAX anfragen sendet um das Attribut *opened* der Tabelle *options* zu aktualisieren, den Button auszutauschen (wenn beendet wurde, muss der öffnen Knopf geladen werden) und das Statusfeld, welches anzeigt ob die Wahl gerade geschlossen oder geöffnet ist, zu aktualisieren. Dies wird durch die PHP Methoden *toggleOpened1()* und *toggleOpened2()* umgesetzt. Diese Umsetzung ist sehr langsam, da jeweils die HTML Zeile und der Button vom Server neu geladen werden. Wenn mehr Zeit zur Verfügung stehen würde, würde ich die Implementierung dahingehend ändern, dass die Anzeige alleine mit JS realisiert wird, und das Klicken des Buttons jeweils nur den Befehl zum Ändern des Datenbank Attributs auslöst, um die Performanz zu verbessern.

Bei dem Klick auf den „*Zuweisung starten*“ Button, wird dessen *onclick*-Funktion ausgeführt. Diese zeigt dem Administrator, dass die Zuweisung noch nicht gestartet werden darf oder den Algorithmus anstößt, indem mittels POST Request die Funktion *startAlgo()* aufgerufen wird. Nach erfolgreichem Abschluss des Algorithmus, wird die Dashboard Seite neu geladen und ein Modal mit der Information angezeigt, dass die Zuweisung erfolgreich war. Der Algorithmus geht folgendermaßen vor:

1. Hole alle Studenten aus der Datenbank und füge ihnen das Attribut Priorität hinzu
2. Suche alle Ratings mit Präferenz i (initial 10) raus
3. Iteriere über alle Ratings, welche eine Bewertung mit i erhalten haben
4. Suche alle Studenten heraus, welche diese AG mit i bewertet haben
    a. Falls Anzahl der Bewertungen <= Plätze
        i. Den Studenten die zugehörige AG zuweisen (DB Attribut schreiben + Student aus Algorithmus löschen + Plätze der AG -1)
    b. Ansonsten:
        i. Studenten mit höchster Priorität zuweisen. (Das Attribut Priorität wird jedes Mal inkrementiert, wenn ein Student eine AG belegen will, dies aber nicht möglich ist, weil zu viele Studenten die AG bewertet haben)

   ii. Wenn alle Studenten der höchsten Prio zugewiesen wurden, jedoch noch Studenten sowie Plätze übrig sind, Priorität um eins verringern
   iii. Bei allen Studenten, welche nicht zugewiesen werden konnten, die Priorität um 1 erhöhen.
5. Kehre zu Schritt 2 zurück, wobei i um 1 dekrementiert wird.
6. Der Algorithmus schließt ab, sobald alle Studenten vergeben worden sind.

Anmerkung: Das Attribut *Priorität* ist wahrscheinlich überflüssig, da in der derzeitigen Version, die Studenten nicht frei Bewerten können, sondern einfach die Arbeitsgruppen in eine Reihenfolge bringen sollen. Dementsprechend sieht jede Bewertung gleich aus. Die Prioritäten 2-10 sind jeweils 1 Mal vergeben, alle anderen Arbeitsgruppen sind mit 1 bewertet. Da dies eventuell später noch geändert wird, verändere ich den Algorithmus erst einmal noch nicht. Hätte ich mehr Zeit gehabt, hätte ich das Problem nicht mit einem selber geschriebenen numerischen Algorithmus gelöst, sondern mit Hilfe des Simplexalgorithmus als einfaches Zuordnungsproblem. Dies hätte wahrscheinlich zu besseren und zuverlässigeren Lösungen geführt.

Nachdem die Zuweisung erfolgt ist, wird der Browser beim Betätigen des „*Ergebnisse downloaden*“ Buttons, dazu gezwungen, die dynamisch erzeugte Ergebnis Datei vom Server herunterzuladen. Dies geschieht, indem ein GET Request an */admin_download_results* gesendet wird. Dies löst die Funktion *downloadResultsXlsx()* aus, welche die externe Excel Facade von maatwebsite nutzt. Dies ermöglicht es die Laravel View partials.admin_Ergebnisse direkt in einen Excel Sheet umzuwandeln. Dabei ist vor allem die Zeile

- $sheet->loadView('partials.admin_Ergebnisse', $parameters);

von Bedeutung. Die Parameter wurden vor übergabe berechnet. Es werden zum Beispiel die zugewiesenen Studenten und die gewählten Arbeitsgruppen ausgegeben, wobei die Restplätze der Arbeitsgruppen direkt berechnet werden. Zusätzlich werden noch Statistiken zu den Arbeitsgruppen erstellt.

Der „*Begrüßungstext ändern*“ Button löst das Öffnen des *modals.acp-begruessungstext* auf. In dieses wird, je nachdem welche Sprache derzeit ausgewählt ist, der deutsche oder der englische Willkommenstext aus der Datenbank geladen. Wird hier eine Änderung getätigt, wird die PHP-Methode *saveWelcome()* aufgerufen. Diese aktualisiert den gerade veränderten Text.

Durch den Button „Wahlgang beenden“ können Daten bequem aus der Datenbank gelöscht werden. Dazu wird das *modals.acp-end-election* aufgerufen, dort kann der Administrator entweder alle Studenten und Ratings zu löschen, nur die Ratings zu löschen, oder nur die zugewiesenen Arbeitsgruppen von den Studenten zu entfernen. Um sicherzugehen, dass der Administrator dies wirklich möchte, muss das Löschen durch Eingabe des Administratoren Passworts bestätigt werden. Je nachdem was gelöscht werden soll, wird eine der Funktionen *checkAdmin()*, *deleteRatings()* oder *deleteAssignments()* aufgerufen. Das Modal enthält dabei jeweils Formulare für das Löschen, um dynamisch überprüfen zu können, ob das Passwort übereinstimmt, wurde jeweils das *submit*-event mit JavaScript überschrieben. Das Überprüfen des Passwortes findet im Controller mit Hilfe der *check()* Funktion der Hash Facade statt. Diese nutzt den gleichen Verschlüsselungsalgorithmus, welcher auch beim Registrieren genutzt wird.

Admin AG Übersicht

Unter der URL */admin_AG* ist eine Übersicht aller Arbeitsgruppen zu finden. Dabei wird initial die Methode *showGroups()* des *admin/workgroupControllers* aufgerufen. Dieser lädt alle zugehörigen Daten aus der Datenbank und übergibt sie der View *admin_AG*. Nehmen wir an zu Beginn wäre die Datenbank leer und es gäbe noch keine Arbeitsgruppen. Dann kann man durch das klicken auf den „hinzufügen“ Button, die JavaScript Funktion *anhaengen()* aufrufen, welche eine neue Zeile in die Tabelle einfügt. Also es wird erstmal nur der DOM Baum manipuliert. Sobald ein Student eine Bewertung abgegeben hat, dürfen keine neuen AGs mehr hinzugefügt werden, da diese Bewertung ansonsten unvollständig wäre, dies wird in der View *admin_AG* realisiert, indem das mitgegebene Attribut *numberGroups* auf *>0* überprüft wird.

Die neue Zeile ist erstmal leer und besteht aus vier Input Feldern und einem Button zum Löschen der Zeile. Nachdem die gewünschte Anzahl AGs erstellt und befüllt wurde, muss der „speichern“ Knopf betätigt werden. Dieser ruft die JavaScript Funktion *checkSave()* auf, welche überprüft, ob die eingegebenen Daten für die Datenbank konsistent sind, das bedeutet, die Spalten Gruppenname, Gruppenleiter und Plätze muss für jede AG befüllt sein und die Gruppennamen müssen eindeutig sein. Ist dies nicht der Fall, wird eine entsprechende Meldung (Bootstrap alert) ausgegeben. Wenn der aktuelle Stand des DOM valide ist, wird das AG_form abgeschickt, wobei das submit event überschrieben wurde. Der neue Submit-Handler erhält ein JSON, welches die IDs der neu hinzugefügten AGs enthält, diese werden über ihren Namen gesucht und die ID in die entsprechende hidden Zelle eingefügt. Das Speichern der AGs wird im Controller über die Funktion *saveGroups()* realisiert. Dieser liest die übergebenen Arrays der Ids, Namen, Leiter, Spots und Zeitpunkten aus, updatet die schon vorhandenen AGs und fügt die neu hinzugefügten in die Datenbank ein. Die Überprüfung, ob eine AG bereits in der Datenbank ist, wird durch das Prüfen der ID auf NULL umgesetzt, neu hinzugefügte AGS haben im DOM noch keine ID.

Soll eine AG wieder gelöscht werden, kann dies durch den löschButton der jeweiligen Zeile umgesetzt werden. Wird einer dieser löschButtons geklickt, wird der HTML Button per JS in der Variable trigger gespeichert und ein Modal geöffnet, in das die Daten der zu löschenden AG geladen werden, um zu bestätigen, dass diese AG wirklich gelöscht werden soll. Dabei werden die Daten anhand des gespeicherten triggers gesucht, alle ID Input Felder besitzen die Klasse „id“, alle Gruppenleiter die Klasse „gl“ etc. Durch die JQuery Funktion *row.find(„input.id“).val()* kann der jeweilige Wert ausgelesen werden. Wenn in diesem Modal das Löschen bestätigt wird, wird die JS Funktion *deleteTrigger()* aufgerufen. Diese Überprüft, ob die AG eine ID besitzt, sollte dies nicht der Fall sein, wurde sie gerade erst hinzugefügt, und kann einfach aus dem DOM entfernt werden. Andernfalls wird die Funktion *deleteGroup()* des *admin/workgroupController* aufgerufen. Falls dieser AG bereits Studenten hinzugefügt wurde, wird eine SQLException geworfen, was eine Benachrichtigung des Admins zur Folge hat, in dem Fall darf die AG nicht gelöscht werden. Ansonsten wird die AG und alle Ratings dieser AG aus der Datenbank entfernt und die AG Tabelle durch die View *ajax.admin_AG_table* aktualisiert.

Da es eventuell eine ganze Menge an AGs geben kann, haben wir eine Suchfunktion implementiert. Wird der SuchButton betätigt oder im Suchfeld Enter gedrückt, wird eine AJAX anfrage geschickt und die *searchGroups()* Methode des Controllers aufgerufen. Dabei werden in der Datenbank alle Attribute der AGs auf Übereinstimmung mit der Suchanfrage überprüft, und die AG_table mit den passenden AGs aktualisiert.

Admin Studenten Übersicht

Durch das Aufrufen der URL */admin_studenten*, wird die Funktion *showStudents()* des *admin/studentController* aufgerufen, welche mit Parametern die View admin_studenten aufruft. Je nachdem, ob die Zuweisung gestartet wurde (*ob sizeof($zugewieseneStudenten) ==0*), werden entweder nur die Daten der Studenten in die Tabelle geladen, oder die Studenten werden mit der zugewiesenen Gruppe und der zugehörigen Bewertung verknüpft. Die zweite Abfrage muss dabei als *outer join* ausgeführt werden, da es theoretisch die Möglichkeit geben kann, dass sich Studenten nachdem die Zuweisung gestartet wurde, noch anmelden.

Da es auch hierbei wieder dazu kommen kann, dass sich eine Großzahl an Studenten anmeldet, haben wir auch hier wieder eine Suchfunktion implementiert. Diese funktioniert identisch zu der AG-Suchfunktion. Beim Drücken der Enter Taste oder des Buttons, wird eine Suchanfrage an die *searchStudents()* Methode gesendet, welche die Tabelle, mit den der Suchanfrage entsprechenden Studenten, aktualisiert. Dabei wird die *view ajax.admin_Studenten_table* mit den gleichen Parametern, wie bei dem Laden der Seite aufgerufen, und in die Tabelle geladen.

Auch das Löschen einer AG funktioniert ähnlich, wie bei dem Löschen eines Studenten. Der entsprechende löschStudentButton wird beim Klicken gespeichert, und es wird ein Modal mit den Daten des zu löschenden Studenten aufgerufen. Wird der Lösch-Button im Modal bestätigt, so wird die JS Funktion *deleteStudentTrigger()* ausgeführt, und es wird ein http Request an die *deleteStudent()* Methode gesendet, wobei die ID des zu löschenden Studenten übergeben wird. Den http Request habe ich ein wenig unschön implementiert, indem ich kein AJAX nutze, sondern die Seite nach dem Löschen neu lade, bzw ein redirect an */admin_studenten* durchführe, dies ist der frühen Implementierung dieser Funktion zuzurechnen.

Wird ein bearbeiten-Button gedrückt, so wird per JS ein Redirect zu */admin_studenten_bearbeiten* durchgeführt, wobei die ID des zu bearbeitenden Studenten mitgegeben wird.

Admin Student Bearbeiten

Beim Aufrufen der URL */admin_studenten_bearbeiten*, wird die Methode *editStudent()* des *admin/studentController* aufgerufen, dabei wird der Student anhand der übergebenen ID aus der Datenbank gezogen und die View *admin_studenten_bearbeiten* mit dessen Daten als Parameter übergeben. Dabei wird die Zeit der Letzten Anmeldung und Registrierung ebenfalls aus der Datenbank geholt, und per *preg_replace* um 2 Stunden erhöht, da für die Speicherung mit einer anderen Zeitzone gerechnet wird. Zusätzlich wird überprüft, ob der Student bereits seine Wahl abgegeben hat, ist dies der Fall, so werden alle seine Ratings ebenfalls als Parameter übergeben. In der View wird dann der übergebene Parameter überprüft und je nachdem können die Ratings durch das betätigen eines Buttons aufgerufen und sogar verändert werden.

Die Daten des Studenten werden jeweils in Input Felder geladen und können dort geändert werden. Durch das Betätigen des „Änderung speichern“ Buttons, wird das Formular abgeschickt, wobei das submit Event wieder überschrieben wurde. Das Formular wird an die php Methode *saveStudent()* geschickt, dort wird der Request in ein studendModel ausgelesen. Dies ist das einzige Mal, dass im AdminBereich ein Model genutzt wird, da mir dies redundant erschien, da ich mit den Daten nicht großartig auswerte, sondern immer direkt wieder in die Datenbank zurückschreibe. Im nächsten Project würde ich Models allerdings in Kombination mit eloquent häufiger nutzen. Die geänderten

Daten des Studenten werden zunächst im Controller überprüft, ob die neue Matrikelnummer und Email unique ist. Sollte dies nicht der Fall sein, wird ein JSON zurückgegeben und per JS ausgewertet um dem Administrator eine Rückmeldung zu gegeben. Zusätzlich werden die Änderungen des Formulars zurückgesetzt. Sind die eingegebenen Daten korrekt, wird der entsprechende Eintrag in der Datenbank aktualisiert und der Admin wird auf */admin_studenten* zurückgeleitet. Durch einen Klick auf „Abbrechen“, wird der Nutzer ebenfalls auf */admin_studenten* zurückgeleitet, ohne die Änderungen zu speichern.

Falls der zu bearbeitende Student bereits eine Wahl getroffen hat, besteht die Möglichkeit auf den Button „Bewertungen einsehen und bearbeiten“ zu klicken, was dazu führt, dass das modal acp-stud-wahl geöffnet wird. Dieses enthält alle abgegebenen Bewertungen des Studenten, wobei die Note der Bewertung verändert werden kann. Wird der „Änderungen speichern“ Button geklickt, wird die JS Funktion *validateRating()* aufgerufen, welche überprüft, ob alle Noten ganzzahlig und zwischen 1 und 10 sind, ist dies nicht der Fall wird eine entsprechende Meldung ausgegeben. Ansonsten wird das Formular abgeschickt, wobei die PHP-Methode *saveRating()* aufgerufen wird. Dadurch werden alle Ratings des Studenten in der Datenbank aktualisiert und der neue Durchschnittswert auf der Seite eingetragen.

Das Betätigen des reset-Buttons lädt die gespeicherten Ratings des Studenten neu, um alle getätigten Änderungen rückgängig zu machen. Dies wird durch den Aufruf der JS Funktion *resetRating()* realisiert, welche wiederum per AJAX Request die PHP-Methode *getRating()* des Controllers auslöst. Hier wird die *view ajax.admin_Rating_table* mit den aktuellen Ratings aus der Datenbank in die Rating-Tabelle geladen.

User-Bereich

Das Dashboard

Dem Benutzer wird über das Dashboard Auskunft über Wahlstatus und Wahlbewertungen geliefert. Ihm wird die Möglichkeit gegeben die Seite durch ein persönliches Benutzerbild zu personalisieren und seine Angaben einzusehen.

Wurde der Wahlvorgang durch ein Mitglied der Fachschaft noch nicht geschlossen, so hat dieser die Möglichkeit seine abgegebene Bewertung noch einmal zu überdenken und gegebenenfalls zu ändern. Wurde der Wahlvorgang geschlossen, so sieht der Anwender anstelle des Links zur Wahl eine Information darüber.

Profil bearbeiten gehört ebenfalls zu den Funktionalitäten die nur solange zu erreichen sind, wie dass der Wahlvorgang noch nicht beendet wurde.

Der Anwender kann das Dashboard direkt nach Login einsehen. Bereitgestellt werden die Daten durch den *UserController*, welcher die Daten an das *dashboard* Template weitergibt.

Das *dashboard* Template erweitert ebenfalls das strukturelle Template *app*.

Profil bearbeiten

In dieser Ansicht kann der Benutzer seine Daten, falls nicht korrekt, bearbeiten.

Hierbei ist zu beachten, dass Matrikelnummer und Email Adresse in der Datenbank *unique* sind. Aus diesme Grund wird vor speichern der geänderten Daten geprüft, ob dieser Constraint nicht verletzt

wird. Hierzu wurde eine eigene Validationsregel implementiert. Implementiert wurde diese in der boot-Methode der AppServiceProvider-Klasse (*app\Provider\AppServiceProvider*).

In dieser Regel wird geprüft, ob die übergebenen Werte zum einen in der Tabelle einzigartig sind oder ob die eingegebenen Werte bereits zu dem Anwender gehören. So kann ein Anwender, für den bereits die Matrikelnummer 123456 hinterlegt ist, diese wieder eingeben. Macht dies nun ein anderer Benutzer mit dieser Matrikelnummer, erhält dieser eine Fehlernachricht.

Sollte ein Anwender das Zugangspasswort ändern wollen, so muss das aktuelle Passwort mitgesendet werden. Ist dies nicht der Fall, so wird der Anwender mit Hilfe des *Validators* (mitgeliefert von Laravel) zurück geleitet mit entsprechenden Fehlermeldungen. Stimmt das neue Passwort mit dem Bestätigungspasswort nicht überein schlägt auch hier der Validator fehl und leitet den Anwender zurück.

Die Wahl

Bereits bei betreten der Wahl wird dem Anwender anhand kurzer Punkte die Anwendung der Wahl erklärt. Hierbei ist zu beachten, dass es sich nicht nur um die Generelle Nutzung dreht, sondern auch wie der Anwender seine Bewertung der Arbeitsgruppen abgibt.

Der Anwender kann durch manuelles Sortieren der Arbeitsgruppen diesen eine persönliche Gewichtung geben. Selbstverständlich kann der Anwender, falls er mit dem Ergebnis nicht zufrieden ist, dies auch wieder in die ursprüngliche Sortierung bringen. Sobald der Anwender seine Bewertung abgegeben hat und mit der Reihenfolge zufrieden ist, kann diese gespeichert werden.

Die visuelle Darstellung der Wahl wird innerhalb des wahl-Templates übernommen. Dieses Template erweitert ebenfalls das Template app. Im Gegenzug zu anderen Templates wird hier unter anderem jQuery UI sowie jQuery UI TouchPunch eingebunden sowie Seitenspezifisches JavaScript und CSS. TouchPunch wurde eingebunden um die Wahl auch auf mobilen Endgeräten zu ermöglichen.

Das Speichern übernimmt der *RatingController*. Hier werden die Daten über einen HTTP POST-Request gesendet. Durch die Sortierung der Arbeitsgruppen erhält jede Arbeitsgruppe eine spezielle Gewichtung (*siehe Drag and Drop Wahlliste*).

Footer

Der Footer ist ein vom eigentlichen Inhalt getrennter Bereich. In dieser Web-Applikation wird er durch einen dunklen Hintergrund hervorgehoben. Inhalt dieses Footers sind diverse Social Media Links, ein Link zu der offiziellen Fachschaftsseite und ein Link für die Kontaktaufnahme mit der Fachschaft Biologie Konstanz. Weiterhin kann der Anwender eine kurze Information über die Fachschaft Biologie Konstanz sowie die Postanschrift finden.

Der Footer gilt als strukturelles Template. Er wird innerhalb der beiden Templates app und admin eingebunden und tritt dementsprechend auf jeder Seite der Applikation auf.

Besondere Charakteristika der Web-Anwendung

Bilingualität

Aufgrund der internationalen Stellung der Universität Konstanz stand fest, dass die Anwendung nicht nur für deutsche Studierende erreichbar sein soll, sondern auch für internationale Studierende.

Dies wurde durch die Implementierung einer Before-Middleware (app/http/Middleware/*Language*) erreicht. Da die Sprache nach dem Wechseln wieder auf die Default Sprache umstellte, musste ein Weg gefunden werden die Sprache dauerhaft zu wechseln. In dieser Middleware wird bei jedem Seitenaufruf geprüft ob eine session-variable mit dem key „*locale*“ gesetzt ist. Ist dies der Fall so wird in der Anwendung diese locale gesetzt. Bei einem Aufruf einer Route werden alle dieser Route zugewiesenen Middlewares der Reihe nach aufgerufen. Hierbei spielt die Reihenfolge eine entscheidende Rolle. Eine Before-Middleware wird vor Aufrufen der eigentlichen Methode aufgerufen. Gemäß der offiziellen Dokumentation kann man sich eine Middleware wie eine Art Schicht vorstellen durch die ein HTTP-Request gehen muss. Hier werden dann alle notwendigen Einstellungen, wie eben hier die Sprachauswahl, gesetzt. Der Ablauf innerhalb der Web-Applikation ist, dass ein Anwender die Sprache über das Dropdown-Menü in der Navigation auswählt. Nach erfolgreichem setzen der geänderten Sprache wird der Benutzer wieder zurückgeleitet. Für diese Funktionalität spielt eine minimale Controllerfunktion innerhalb der web.php eine Rolle. Diese leitet den Anwender zurück.

Drag and Drop Wahlliste

Die Studierenden können ihre Bewertungen für die zur Verfügung stehenden Kurse mit Hilfe einer Drag and Drop-Liste bewerten. Hierbei kann ein Benutzer lediglich die Sortierung der Arbeitsgruppen vornehmen, die Bewertung wird dann für die erste Arbeitsgruppe mit 10 Punkten, für die zweite Arbeitsgruppe mit 9 Punkten bewertet, bis wir bei nur noch einem Punkt angekommen sind. Ab dieser Arbeitsgruppe erhalten alle weiteren Arbeitsgruppen nur noch einen Punkt.

Im ersten Implementierungsansatz wurde mit Inputranges gearbeitet, dies war aufwendig, da wir sowohl für die Desktopansicht als auch die mobile Ansicht je eine Inputrange benötigt haben. Hierdurch kamen wir auf eine gesamte Anzahl von 4 Inputfeldern, die für eine Arbeitsgruppe notwendig waren. Hinzukam die Problematik, dass wenn ein Benutzer seine Wahl ändern wollte, die Bewertung für eine Arbeitsgruppe jedoch nicht geändert hat, diese in der Änderung mit 0 Punkten bewertet wurde. So kam es zu ungewollten Veränderungen an der Bewertung. Eine einfachere Alternative hat somit die sortablelist von jQuery UI geliefert. Hier wird jeder Arbeitsgruppe ein Inputfeld vom typ *hidden* mitgeliefert. Da der Name des Feldes für jede Arbeitsgruppe gleich ist und mit eckigen Klammern beendet wird (*name=“input[]“*), wird ein Array an den Controller geliefert der die entsprechend sortierten Elemente verarbeiten kann.

Somit entfiel das anwenden von Überprüfungen, ob ein Benutzer die Bewertung gemäß bestimmter Kriterien erfolgte oder nicht.

Die Drag and Drop Wahlliste wurde innerhalb des Blade-Templates *wahl* implementiert. Dieses Template erweitert das strukturelle Template *app* (siehe Struktur). Das speichern und verarbeiten dieser Bewertungen übernimmt der eigens dafür geschriebene *RatingController*.

Dynamischer Bilderupload

Ein weiteres Merkmal der Web-Anwendung ist der dynamische Bilderupload, bei dem der Anwender Bilder über Drag & Drop oder über das FileSystem hochladen kann. In beiden Fällen werden die Bilder direkt nach der Auswahl über einen AJAX Request an die hinterlegte Route gesendet. Diese wird vom UserController, welche alle Aktionen des Benutzers steuert, verarbeitet und für den angemeldeten Benutzer zur Verfügung gestellt. Dieser Dynamische Bilderupload wird innerhalb des Dashboards implementiert und ist auch nur hier erreichbar. Seitenspezifisches Javascript ermöglicht die Darstellung der „Dropzone“, welche an ein Form-Tag gebunden ist. Gefunden werden kann dieser Code innerhalb der *resources/views/modals/crop*.

Komprimierung von JavaScript und Cascading Stylesheets

Da es sich bei der Web-Anwendung um eine Anwendung für mobile und Desktop Systeme handelt, spielt der Datenverkehr eine große Rolle. In der Regel verwenden Anwender, die die Web-Applikation von Unterwegs aus aufrufen ihr Datenvolumen. Die Grenzen für den mobilen Datenverkehr sind meist relativ gering, aus diesem Grund haben wir uns dazu entschieden die *Cascading Stylesheets* und *JavaScript-Dateien* zu komprimieren. Verwendete Bibliotheken dritter wurden, falls möglich über ein *Content Delivery Network* geladen. Sollten wir solche Bibliotheken einbinden, so haben wir diese nur dann komprimiert, wenn dies ohne Funktionsverlust möglich war.

Um diese Dateien zu komprimieren wurden die für Node.JS verfügbaren Module uglifyjs sowie lessc und lessc-clean-css verwendet. Bei ersterem handelt es sich um ein reines Modul um JavaScript-Code zu komprimieren. Bei Lessc handelt es sich um einen von Less bereit gestelltes Compiler-modul, welches nicht nur die Komprimierung von CSS ermöglicht, sondern auch das transpilieren von less-Syntax in äquivalentes css-Syntax.

Nutzung von JavaScript / Cascading Stylesheet Libraries von dritten

Um der Anwendung einen modernen Look and Feel zu verleihen werden die modernsten JavaScript und CSS Libraries verwendet. Darunter ist nicht nur jQuery in der aktuellsten Version (v.3.2.1, Stand 30.06.2017), sondern auch Bootstrap in Version 3.3.7. Das verwenden dieser modernen Libraries ermöglicht es uns Code effizient und sauber zu schreiben ohne größere Probleme.

Die Library Bootstrap ermöglicht uns neben dem Look and Feel die Portierung der Web-Anwendung in eine dynamische, mobile first Web-Applikation. Hier wird besonderen Wert auf die Dynamik der Elemente gelegt die auf einem mobilen Endgerät eine andere Positionierung, Größe, etc. haben können, als auf einem Desktop PC.

Da wir so nur das grundlegende abdecken, haben wir uns für den Bilder Upload für die Library Dropzone entschieden. Diese ermöglicht es uns bequem über AJAX Bilder hoch zu laden. Es ermöglicht uns außerdem das Drap & Drop von Bildern. Hinzu kommt ein modernes Erscheinungsbild des Upload Elements.

Die Liste für die Bewertungen wurde über jQuery UI realisiert. Hierbei kommt die implementierte sortabelList zum Einsatz. Sie ermöglicht es uns über Drag & Drop Elemente so zu positionieren, wie wir es möchten.

Kontaktaufnahme mit der Fachschaft Biologie der Universtität Konstanz

Es wird dem Benutzer ermöglicht über ein Formular direkt Kontakt mit der Fachschaft Konstanz aufzunehmen. Hierbei kommt die von Laravel mitgelieferte Mailable zum Einsatz. In Kombination mit einem gültigen E-Mail Account und SMTP-Zugang können wir E-Mails über unsere Web-Anwendung versenden.

In diesem Formular werden alle Felder benötigt. Sollte ein Feld fehlen, wird über ein Validator eine entsprechende Fehlermeldung zurückgeworfen.

Das sich beim Klick auf den Link „E-Mail schreiben“ öffnende Formularfenster wird über einen AJAX-Request versendet. Dieses Fenster wird nun so manipuliert, dass der Benutzer sieht, dass eine Bearbeitung erfolgt. Bei erfolgreichem Versenden erhält der Benutzer die Information darüber. Bei fehlerhaften oder fehlenden Angaben wird der Anwender auf das Formular zurück geleitet mit Ausgabe der entsprechenden Fehlermeldungen. Das Kontaktformular für den Mailverkehr wird über mehrere Templates zusammengesetzt. Das Modal, also das eigentliche Fenster, innerhalb der *resources/views/modals/mail* zu finden. Dieses Template bindet via @include-Direktive das Formular *resources/view/mail/forms/contact* ein. Verarbeitet und versenden der Kontakt E-Mail übernimmt der MailController.

Verwendete Technologie: Git

Neben den Basis-Technologien, wie HTML5, JavaScript und CSS3 wurde die Versionverwaltungsoberfläche Github verwendet. Im Verlauf der Projektentwicklung wurden 435 Commits bei 95 Issues durchgeführt.

Wir haben das komplette Projekt in Issues aufgeteilt, so dass zu jeder Zeit jeder genau wusste was und wann dies zu tun ist. Wie in jedem Projekt, welches über eine Versionsverwaltung entwickelt wird haben auch wir mehrfach Mergeprobleme erhalten. Diese konnten mit jedem erneuten Auftreten schneller und effizienter behoben werden, so dass am Ende keine Konflikte mehr entstanden sind.

Für genauere Einsicht kann folgendem Link gefolgt werden: https://github.com/BioThemenVergabe