

Praktikumsbericht swop

Brigitte Kwasny, Juliane Richter, Lasse Weinbrandt, Lukas Uzolas, Patricia Häußer

Konzeption und Zielsetzung	2
Idee und Problemstellung	2
Konzept	2
Technologien	4
Ziele	5
Rollen	5
Zeitliche Planung	6
Arbeitspakete	6
Frontend	7
Navigation	7
Dashboard	8
Tauschanfragen-Dialog	9
Settings	10
Registrieren, Login, Passwort vergessen	10
Backend	11
Match-Algorithmus	12
Prozess der SwopCard-Erstellung	12
Schnittstelle	13
Kritik an unserer Schnittstelle	14
Fazit	14
Anhang	16
/	1

Konzeption und Zielsetzung

Unter dem Projekttitel swop entwickeln wir einen Marktplatz, auf dem Studenten untereinander Plätze in Veranstaltungen tauschen können.

Bei der Findung einer Anwendungsidee für die Praktikumsarbeit einigten wir uns im Team darauf, eine Anwendung entwickeln zu wollen, mit deren Problemstellung und Lösungsansatz wir vertraut sind. Die Problemstellung sollte also von Relevanz und nicht konstruiert sein. Außerdem war unser Anspruch, mit der Anwendung einen echten Gebrauchswert zu schaffen. Das reine Nachbauen einer schon vorhandenen Anwendung kam daher nicht in Frage.

Idee und Problemstellung

Die Idee für swop ging aus unseren persönlichen Erfahrungen mit dem enormen Organisationsaufwand während der Phase zur Modulwahl zu Beginn jeden Semesters hervor. Nach der Wahl und der Priorisierung der Veranstaltungen und Veranstaltungsgruppen über STiNE kann nicht sichergestellt werden, dass sich keine der zugeordneten Veranstaltungen und Gruppen überschneiden. Ein Wechsel der Gruppe oder Veranstaltung über STiNE ist meist mit dem Risiko verbunden, dass alle anderen Plätze im Modul schon belegt sind, man beim Versuch die Veranstaltung zu wechseln auch seinen eigenen Platz verliert und letztendlich nicht zum Modul eingetragen ist.

Alternativ ist es in den meisten Veranstaltungen möglich, mit einem direkten Tauschpartner nach Einverständnis des Übungs- oder Seminarleiters zu wechseln. Damit wird das oben genannte Problem beim Wechsel einer Veranstaltung über das Studieninformationsnetzwerk umgangen und genau hier setzt die Anwendung swop an. swop deckt diesen Prozess von der Suche eines Tauschpartners bis zur Kontaktaufnahme ab.

Konzept

Zu Beginn der Praktikumsarbeit stand also die Analyse der uns zur Verfügung stehende Daten und Informationen aus STiNE und Studieninformationsnetzwerken anderer Universitäten auf den Aufbau und die Struktur einer im System eingetragenen Veranstaltung. Diese Informationen haben wir zum einen verwendet, um die Speicherung der Kurse in der Datenbank an diese Struktur anzugleichen. Zum anderen erleichtern wir durch die Einheitlichkeit der Benennung der Kurse in STiNE und in unserer Anwendung für den Nutzer das Erstellen und Durchsuchen von Veranstaltungen über swop.

Über den Dialog zur Erstellung von neuen Tauschanfragen (im Folgenden "Tauschkarten" genannt) werden die Nutzer dazu aufgefordert zwei spezifische Veranstaltung für den Tausch festzulegen - eine Biete- und eine Suche-Veranstaltung. Zu Beginn soll dies neben den Profildaten der Nutzer der einzige Weg sein über den Daten in die Datenbank zu gelangen. Parallel dazu haben wir trotzdem eine Liste von Veranstaltungen bei STiNE

beantragt. Da wir uns jedoch nicht auf die Universität Hamburg festlegen wollten, sind wir vorerst beim Modell des reinen User Generated Content geblieben.

Zu Beginn der Konzeptionsphase war eine Strukturierung der Veranstaltungs-Objekte in der Datenbank ("courses") anhand drei Attribute angedacht: Typ (Handelt es sich um eine Veranstaltung in Form einer Vorlesung, einer Übung, eines Seminars, eines Praktikums oder eines Projektes?), Modulname (Welchem Modul ist die Veranstaltung zuzuordnen?) und optional Gruppe (Spezifisch für Seminare oder Übungen, da diese in STiNE mit Zahlen oder Buchstaben aufgeteilt sind.). Dadurch hätte sich eine Veranstaltungs-Bezeichnung vom Aufbau <Typ> <Modulname> <Gruppe> ergeben (Beispiel: Übung Softwareentwicklung 2 Gruppe 07). Ein Wechsel wäre nach dieser Struktur nur in eine Veranstaltung mit demselben Typ und Modulnamen möglich gewesen.

Nach wiederholtem Studieren der Datenstruktur in STiNE, vor allem in anderen Fachbereichen, fiel uns jedoch auf, dass ein Wechsel zwischen Veranstaltungen wider des anfänglichen Konzeptes auch zwischen unterschiedlichen Modulen möglich und gängig ist. Nach Überdenken einiger Randfälle des Konzeptes entschlossen wir uns dazu, ein Veranstaltungs-Objekt über seine in STiNE eingetragene ID zu identifizieren und den Typ und den Namen in der Bezeichnung zusammenzufassen. Diese Struktur spiegelt sich nun in der Datenbank, aber auch bei der Erstellung und Suche einer Veranstaltung in der Benutzeroberfläche wider. Beim Durchsuchen aller Veranstaltungen im Dialog zur Erstellung von Tauschkarten werden die Veranstaltungen nach folgendem Schema dargestellt: <Veranstaltungs-ID> – <Veranstaltungsname>. Diese neue Struktur des Objektes ermöglicht es nun auch zwischen Veranstaltungen aus unterschiedlichen Modulen zu wechseln. Dies bedeutet aber auch, dass unmögliche Tauschkarten erstellt werden können (Beispiel: Wechsel von einem Informatikmodul zu einem Philosophiemodul). Dieser Problematik sind wir uns bewusst, jedoch überwiegt hier der Mehrwert der Flexibilität. Dies bedeutet allerdings auch, dass wir den Nutzer im Onboarding-Prozess darauf hinweisen müssen, dass er/sie selbst für die Richtigkeit und Erfüllbarkeit seiner Tauschkarten verantwortlich ist.

Vorerst wird das System für Studenten der Uni Hamburg entwickelt, wobei wir im Hinblick auf den Nutzen für Studenten anderer Universitäten oder Bildungseinrichtungen gerade auch in diesem Aspekt die Flexibilität nicht einschränken wollen.

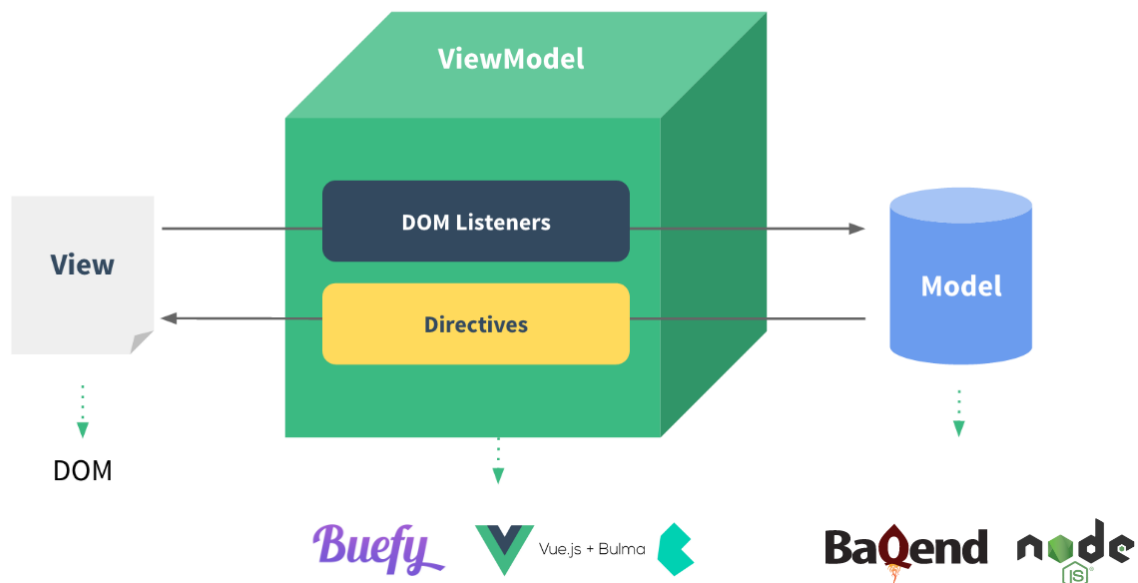
Da wir mit der Anwendung mehrheitlich Digital Natives in unsere Zielgruppe schließen, wählten wir für die Umsetzung des Frontends den Mobile First-Ansatz.

Prototyping

Nach Festhalten der Idee und des Konzeptes verschafften wir uns mit Skizzen und [klickbaren Prototypen](#) (umgesetzt in Adobe Experience Design) unterschiedlicher Szenarien und Customer Journeys einen groben Überblick über den Umfang und die benötigte Datenbankstruktur.

Technologien

Daraus und aus unseren Vorerfahrungen mit unterschiedlichen Tools haben uns für die Verwendung folgender Tools und Frameworks entschieden:



Für die Umsetzung des Backends unserer Anwendung verwenden wir Baqend. In Baqend verwenden wir die ACL (Access Control Lists), um über die Node-Rolle Zugriff auf alle Objekte zu haben. Über eigene Services, die wir in Node-Module aufgeteilt haben, manipulieren wir die Objekte. In der Frontend-Entwicklung arbeiten wir mit dem JavaScript-Framework Vue.js und bedienen uns des CSS-Frameworks Bulma.

Zu Beginn lag unser Anspruch darin, das Backend der Anwendung komplett selbst mit Node.js und MongoDB aufzusetzen und ein RESTful Webservice zu schreiben, um im Projektverlauf den größten Lerngehalt zu erzielen.

Da für die Projektrealisierung nur 3 Wochen angedacht sind, würde eine Realisierung einer RESTful API mehr Zeit in Anspruch nehmen. Daher entschieden für uns in Rücksprache mit dem Praktikumsleiter für Baqend, welches bereits ein User Management, Access Control Lists und eigenen Code für Node-Module zur Verfügung stellt. Nach der Realisierung eines Klassendiagramms (UML, siehe unten) mit entsprechenden Referenzen, realisierten wir dies in Baqend.

Dem Frontend liegt das Javascript-Framework Vue.js zugrunde. Warum haben wir uns für Vue.js und nicht für React, Angular oder jQuery entschieden? Zunächst wussten wir nur, dass es ein gern verwendetes Framework unter Frontend Developern mit Fokus auf UX Design ist. Nach ausführlicherer Recherche überzeugten uns jedoch die Hard Facts des Frameworks: Man kann schnell starten und leicht mit simplem Javascript ohne Einbindung weiterer Plugins oder Bibliotheken wie jQuery reaktive Single-Page-Applications bauen. Zudem

verwendet Vue eine Syntax, die a) klassischem HTML noch näher ist als z.B. JSX bei React und b) die dank guter Dokumentation leicht zu erlernen ist.

Vue bildet eine reaktive View-Komponente und implementiert das Model-View-ViewModel-Modell (MVVM), was einem das Aufsetzen von Controller-Instanzen erspart und leicht zu testen ist. Ein weiterer Vorteil von Vue ist, dass man sein Projekt mittels der schachtelbaren Components modular aufbauen und somit wiederverwendbare UI-Komponenten (Atomic Design) schaffen und diese kombinieren und zusammensetzen kann.

Als Basis für das Styling der UI-Komponenten, entschieden wir uns für Bulma, da es uns moderner und flexibler als Bootstrap erschien und es mit der Library Buefy schon ein Node-Modul gibt, welches Bulma-Komponenten für Vue bereitstellt. Einige Buefy-Elemente ersetzten wir später jedoch wieder durch selbstgeschriebene Bulma-Komponenten ersetzt, da sie nur schwer zu modifizieren sind und uns dadurch in manchen UI-Entscheidungen eingeschränkten.

Ziele

Wie oben erwähnt lag unser Ziel neben dem Meta-Ziel, den Umgang mit einer an eine Datenbank angebotenen Web-Application zu lernen, darin, in den uns zur Verfügung stehenden drei Wochen eine größtenteils funktionsfähige, aber auch aus Nutzersicht angenehm zu bedienende Anwendung zu konzipieren und zu programmieren.

Die geplanten Funktionalitäten ordneten wir daher entweder dem Minimal Viable Product oder dem Backlog zu (siehe dazu die [Präsentationsfolien zur Zwischenpräsentation am 14. August](#)). Das MVP umschließt dabei alle Funktionalitäten, die bis zum offiziellen Ende des Praktikums umgesetzt sein sollten. Die darin definierten Funktionalitäten decken den kompletten Key Customer Journey von der Erstellung einer Tauschkarte über das Verwalten dieser bis hin zum Austausch der Kontaktdaten mit einem Tauschpartner komplett ab. Die im Backlog gelisteten Funktionalitäten erweitern die Anwendungen und den Key Customer Journey und zielen darauf ab, swop auch in unterschiedlichen Kontexten außerhalb der Universität einsetzbar zu machen.

Rollen

Innerhalb des Teams teilen wir uns in zwei Teams auf: Zwei Mitglieder bildeten das Backend-Team, während wir am Frontend der Anwendung zu dritt arbeiteten. Durch die Aufteilung war es leichter, sich gemeinsam in die einzusetzenden Technologien zu erarbeiten, um anschließend innerhalb der Teams einzelne Aufgaben vergeben zu können. Die Aufgaben wurden klar definiert und entsprechend der Kompetenzen der Teammitglieder so aufgeteilt, dass das hoch gesteckte Ziel für die darauffolgenden drei Wochen erreichbar erschien.

Zeitliche Planung

Bei der zeitlichen Planung der Umsetzung der Praktikumsarbeit setzten wir unsere Meilensteine anhand der Termine der Zwischenpräsentationen.

Phase 1 07.08. - 11.08.	Phase 2 14.08. - 18.08.	Phase 3 21.08. - 24.08.
----------------------------	----------------------------	----------------------------

Phase 1:

Einarbeitung der Teams in die verwendeten Technologien

Frontend-Team: Erstellung von Prototypen und ersten statischen HTML-Seiten, derer man sich später bedienen kann

Backend-Team: zunächst Eingelezen in Baqend-Guides, um eine grobe Vorstellung davon zu erhalten, was mit Baqend alles möglich ist, anschließend Aufteilung der Aufgaben und Beginn der Implementation des Code für die Object-Handler und eigene Node-Module

Phase 2:

Frontend-Team: zuerst Implementierung der Anwendung mit Vue.js, Erstellung von Vue Components aus den statischen HTML Files und später Integration der Datenbank

Backend-Team: Fortsetzung der Implementation der Object-Handler und Node-Module

Phase 3:

Während der ersten drei Tage in Phase 3 wurden letzte Implementationen von Funktionalitäten im Frontend und im Backend beendet, damit in den verbleibenden zwei Tagen das MVP getestet und eventuelle Fehler behandelt werden konnten. Zu Beginn der Praktikumsarbeit planten wir in den letzten Tagen noch einige Punkte aus dem Backlog abzarbeiten. Letztendlich entschlossen wir uns jedoch dazu, uns in den letzten Tage noch auf den Feinschliff der Funktionalitäten des MVP und damit der wichtigsten Funktionalitäten zu fokussieren.

Arbeitspakete

Um die einzelnen Aufgaben des Projektes und der Verknüpfung zu organisieren richteten wir am ersten Tag ein gemeinsames [Trello](#)-Board als Task Management-Tool ein. Damit konnten wir uns gegenseitig abzararbeitende Tickets zuweisen ohne den Arbeitsfluss der Anderen zu unterbrechen.

In der Frontend-Entwicklung teilten wir die Aufgabenpakete nach den einzelnen Views der Anwendung auf. Dabei übernahm Lasse das Dashboard und die Navigation/das Routing zwischen den Views, Patricia implementierte den Dialog zur Erstellung einer neuen

Tauschkarte und Juliane setzte die Views zur Nutzerverwaltung (Einstellungen, Login, Registrierung) um.

In der Backend-Entwicklung teilten wir die Aufgaben nach dem UML-Diagramm auf. Brigitte übernahm zunächst die Entwicklung der Handler der `RestrictedUserInfo`, während Lukas sich um die Implementierung der SwopCard-Erstellung kümmerte.

Da wir sukzessiv die einzelnen Arbeitspakete abarbeiteten, kam es an einigen Stellen zu unvorhergesehenen Teilabhängigkeiten zwischen einzelnen Arbeitspaketen, aber auch zwischen Frontend- und Backendfunktionalitäten. Beispielsweise war das Testen einiger Backend-Funktionalitäten erst effizient möglich als die dafür vorgesehene UI fertig implementiert war. Da jedoch in der Zwischenzeit kleinere Aufgabenpakete abgearbeitet werden konnten, kam es durch die Abhängigkeiten nicht zum zeitlichen Verzug des Projektes.

Frontend

Um die Umsetzung des Frontend strukturierter, gleichzeitig aber auch effizienter zu gestalten, wählten wir den Ansatz des [Atomic Designs](#).

Vor der eigentlichen Implementation einigten wir uns daher darauf, die atomaren und molekularen Elemente unseres Interfaces dem CSS-Framework [Bulma](#) zu entnehmen. Die in swop angewandten Elemente sind dem Styleguide unter folgendem Link zu entnehmen: <https://goo.gl/QN98S6>

Navigation

(vue-component Navigation.vue)

Die Navigationsleiste wird jedem eingeloggten Nutzer fixiert am oberen Rand des Viewports angezeigt und bildet die Hauptnavigation auf die verschiedenen Routes und Views. Im Zentrum der Navigationsleiste steht das Logo mit Swoppy - dem Maskottchen der Applikation. Das Logo hat die Funktion eines sogenannten Escape Hatches - also einem UI-Element, welches den User durch Klick zum Ausgangspunkt (hier das Dashboard) zurückführt.

Am rechten Rand der Navigationsleiste befinden sich zwei klickbare Icons, die zu weiteren Views überführen:

1. **Spielregeln** (siehe Spielregeln-View), dargestellt durch einen Rettungsring. Dieser soll den User triggern, bei Unklarheiten hier zu Hilfe suchen.
2. **Settings** (siehe Settings-View), dargestellt durch ein Zahnrad.

Je nachdem, ob der User sich aktuell in einem der o.g. Views befindet, wird dies durch einen kleinen blauen Kreis unterhalb des Icons angezeigt.

Dashboard

(vue-component Dashboard.vue)

Nach dem sich ein User sich eingeloggt oder erfolgreich registriert hat, landet er im Dashboard. Das Dashboard bildet das Zentrum der Applikation und ist Ausgangspunkt aller Handlungen eingeloggter Nutzern. In seiner Kernfunktion ermöglicht das Dashboard es dem User, seine laufenden Tauschkarten und deren Stati einzusehen, diese zu verwalten, den Tausch zu akzeptieren oder abzulehnen oder eine Tauschkarten zu löschen. Außerdem gelangt der Nutzer über das Dashboard zum Dialog zur Erstellung einer Tauschkarte.

Jede laufende Karte besteht aus den Titeln, Untergruppen und IDs der Veranstaltungen und einem Icon, welches den aktuellen Zustand der Karte wiedergibt. Je nach Zustand werden durch Aufklappen der Tauschkarte weitere Informationen (zum Beispiel Kontaktdaten des Tauschpartners) sichtbar. Passt eine Tauschkarte eines Users zu der eines anderen Users, so wird die Karte im Dashboard beider User in der Primärfarbe orange optisch hervorgehoben.

Im unteren Teil des Views befindet sich ein prominent in Sekundärfarbe blau hervorgehobener zentrierter Button, der zur Erstellung einer neuen Tauschkarte führt (Siehe swop-Dialog View). Dieser ist am unteren Teil des Viewports fixiert und in Z-Ebene über den Karten angeordnet, um stets sichtbar zu sein, auch wenn der User durch seine Karten scrollt.

Bei den Zuständen der Tauschkarten unterscheiden wir in zwei Kategorien zwischen Tauschkarten, zu denen schon eine passende Gegenanfrage erstellt wurde ("Match") und Tauschkarten, zu denen noch keine passende Gegenanfrage erstellt wurde (ausstehend).

Solange keine passende Gegenanfrage zu einer Tauschkarte erstellt wurde, wird die Karte im Dashboard nicht hervorgehoben und ist mit einem aus einer Person und einem Fragezeichen zusammengesetzten Icon versehen. Die Message dahinter lautet: "Wir sind noch auf der Suche nach einem passenden Tauschpartner für Deine Anfrage."

Sobald eine passende Gegenanfrage eingegangen ist, wird die Tauschkarte in der Primärfarbe hervorgehoben. Drei unterschiedliche Icons weisen hierbei darauf hin, welcher Schritt für den Nutzer als nächstes folgt.

Solange eine Person neben einem Ausrufezeichen abgebildet ist, muss der Nutzer den Match noch bestätigen. Sobald der Nutzer den Match bestätigt hat, wird ihm neben der eigenen Person (links) eine weitere, noch unausgefüllte Person angezeigt um zu signalisieren, dass bereits ein passender Tauschpartner gefunden wurde, auf dessen Bestätigung der Nutzer zur erfolgreichen Kontaktübermittlung noch warten muss. Erst wenn beide Teilhaber an einem Match diesen bestätigt haben, sehen beide zwei ausgefüllte Personen-Icons mit einem die beiden verbindenden Herz. Ist dieser Status aktiv, so werden beiden Nutzern beim Aufklappen der Tauschkarte im Dashboard die Kontaktdaten angezeigt. Der Nutzer kann einen Match auch ablehnen, tut er dies wird er intern nicht mehr in den Karten-Pool automatisch aufgenommen. Das Icon ist eine ausgefüllte Person mit einem

Kreuz daneben. Die Karte von dem Tauschpartner wird dann anschließend wieder in den Karten-Pool geschmissen und die Karte ist im Dashboard nicht mehr hervorgehoben.

Tauschanfragen-Dialog

(vue-component *SwopDialog.vue*)

Dieses Feature deckt den größten Teil des Key Customer Journeys ab. Hier werden die im Dashboard visualisierten Tauschkarten und neue Veranstaltungen erstellt und in die Datenbank gespeichert. Während der Konzeptionsphase hat sich eben dieser Prozess als weit umfangreicher und fehleranfälliger erwiesen als gedacht. Daher versuchten wir bei der Implementation des Tauschanfragen-Dialogs, dem Nutzer so viel Denkaufwand wie möglich abzunehmen und ihn in kleinen Schritten mittels Stepwise-Navigation durch den Prozess führen.

Das Erstellen einer neuen Tauschkarte lässt sich in drei Schritte unterteilen:

1. Die Angabe der Veranstaltung und eventuell der Untergruppe, aus der man **heraus** wechseln möchte.
2. Die Angaben der Veranstaltungen oder Untergruppen, in die man **hinein** wechseln möchte.
3. Eine Übersicht über die getätigten Eingaben mit abschließender Bestätigung.

Wir unterscheiden hierbei zwischen zwei Arten von Tauschanfragen: i) dem Wechsel aus einer Untergruppe einer Veranstaltung in eine andere Untergruppe derselben Veranstaltung und ii) dem Wechsel aus einer Veranstaltung ohne Untergruppe in eine andere Veranstaltung. Zu Beginn der Konzeptionsphase planten wir dem Nutzer nur einen Wechsel zwischen Untergruppen einer Veranstaltung zu ermöglichen. Bei der Analyse des Aufbaus von Veranstaltungen der Uni Hamburg, vor allem aber in anderen Fachbereichen, stellte sich jedoch heraus, dass es durchaus Szenarien gibt, in denen ein Wechsel zwischen Veranstaltungen mit unterschiedlichen IDs von Nöten ist. Trotz der erhöhten Fehleranfälligkeit beim Tausch zwischen unterschiedlichen Veranstaltungen entschieden wir uns dazu, diesen Fall komplett zu abzudecken. Wir können dadurch zwar nicht sicherstellen, dass über swop unmögliche Tauschanfragen (beispielsweise ein Wechsel zwischen einem Informatik- und einem Philosophiemodul) gestellt werden, schränken den Nutzer und die Funktionen gleichzeitig nicht ein.

In den Spielregeln legen wir daher einen Schwerpunkt auf die Betonung der Eigenverantwortung eines jeden Nutzers der Anwendung swop. Da es sich vorerst bei allen Daten in der swop-Datenbank um User Generated Content handelt – bevor eine Veranstaltung im Dropdown-Menü erscheint und ausgewählt werden kann, muss sie von einem Nutzer der Anwendung erstellt worden sein – und wir einen reinen Marktplatz bieten, basiert ein großer Teil der Anwendung auf die Gemeinschaftlichkeit der Nutzer.

Durch Frontend-Validierungen während und nach jedem abgeschlossenen Schritt verhindern wir, dass eine mögliche Fehlermeldung erst beim Abschicken der Anfrage an die Datenbank

nach der Bestätigung auftritt. Dadurch weiß der Nutzer immer, an welcher Stelle der Fehler auftritt und bleibt der Anwendung wegen des ausbleibenden Frustrationseffektes treu.

Um routinierten swop-Nutzern bei der Erstellung einer Tauschkarte Zeit zu ersparen, ist größtenteils eine Navigation über das Keyboard möglich.

Settings

(vue-component Settings.vue)

Über das Zahnrad in der Navigation gelangt der User zu den Einstellungen. Hier kann er seine Profildaten bearbeiten. Dazu gehören die Änderung des Namens und des Passwortes, sowie das Löschen des Profiles. Das Löschen verfolgt erst nach einer zusätzlichen Bestätigung um Fehler zu minimieren. Die Implementation der Lösch-Funktion ist jedoch noch nicht realisiert, da wir ebenfalls die SwopCards vom User löschen müssten sowie auch die dazugehörigen Matches und die SwopCards der Tauschpartner wieder in den Karten-Pool werfen müssten, um einen reibungslosen Löschvorgang gewährleisten zu können.

Des Weiteren kann der User eine zusätzliche Mail-Adresse für Benachrichtigungen aus der App hinzufügen. Für dieses Feature haben wir uns entschieden, da viele Studenten seltener auf ihre Uni-Mail, als auf ihre private Mail zu greifen, unsere Matches, jedoch schneller Beantwortung bedürfen. Um unsere Anwendung trotzdem vor Missbrauch zu schützen, bleibt die Uni-Mail die Login-Mail und kann nicht geändert werden. Die Uni-Mail kann aber trotzdem auch in den Einstellungen eingesehen werden.

Bei Aufruf einer Funktion (z. B. Änderung des Passwortes) bekommt der User immer ein visuelles Feedback. Somit kann sichergestellt werden, dass der User die Funktionen richtig benutzt und die Fehleranalyse wird für den User leichter gemacht.

Der Settings-View beinhaltet außerdem den Ausstiegspunkt, auf welchem man sich aus der Applikation ausloggen kann.

Registrieren, Login, Passwort vergessen

(vue-component signup.vue, login.vue, ForgetPassword.vue, ResetPassword.vue)

Um unsere Anwendung nutzen zu können, müssen sich alle User ein eigenes Profil erstellen. Dies geschieht über den Sign-Up-Screen

Hier muss der User seinen Vornamen oder anderen bevorzugten Displaynamen angeben, seine Uni-Mailadresse und sein persönliches Passwort. Außerdem muss der User die "Allgemeinen Gurkenbedingungen" akzeptieren. Die "Allgemeinen Gurkenbedingungen" beinhalten die EU Verordnung zur Festsetzung von Qualitätsnormen für Gurken und ersetzen vorerst unsere Allgemeinen Geschäftsbedingungen.

Wenn alle angegebenen Daten den Anforderungen entsprechen, kann sich der User über den Registrieren-Button registrieren. Wenn die Daten fehlerhaft sind, bekommt der User das entsprechende visuelle Feedback.

Über den Button "Bereits Mitglied?" gelangt der User auf den Login-View. Dieser ist immer die erste Seite, die der User sieht, wenn er nicht eingeloggt ist ('/'). Hier kann sich der User mit seinen Benutzerdaten einloggen. Sollte ein User im eingeloggten Zustand per URL auf den Basispfad oder den SignUp-Screen gehen, wird er automatisch ins Dashboard weitergeleitet.

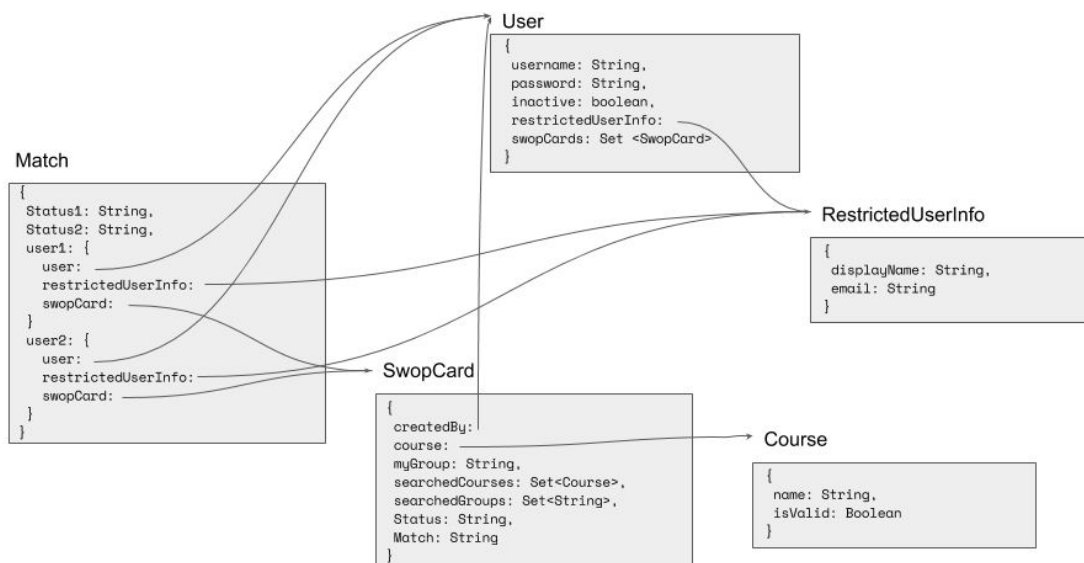
Stimmen die Mail-Adresse und das Passwort mit den Daten in unseren Datenbank überein, gelangt er über den Button "Einloggen" zu seinem Dashboard (siehe Dashboard View). Sind die Daten nicht korrekt erhält der User das entsprechende visuelle Feedback.

Über den Button "Vergessen?" am Passwort-Feld kann der User ein vergessenes Passwort neu setzen. Hierfür gibt der User im Passwort-vergessen View seine Mail-Adresse an. Über einen Link, welcher der User per Mail bekommt, könnte der User auf den Passwort-reset View gelangen und kann dort ein neues Passwort setzen. Aus Zeitgründen haben wir diese Funktion aber nur im Frontend implementiert. Das serverseitige Versenden eines Reset-Links per Mail ist noch nicht vorhanden.

Das Design der Signup- und Login-Screens unterscheidet sich außerdem von denen eines eingeloggten Users. Der Hintergrund ist hier in unserer Primärfarbe (Orange) eingefärbt. Der Container, welcher den Inhalt umfasst, ist aber weiterhin weiß und hebt sich somit vom Hintergrund ab.

Backend

Im Backend bestand die Herausforderung aus den Prototypes und Wireframes entsprechende Klassen bzw. Objekte zu abstrahieren und zu modellieren. Mit ersten Zeichnungen eines Entity-Relationship-Models kamen wir so unseren Klassen näher, die im folgendem Klassendiagramm finalisiert zu sehen sind:



Das User-Schema besteht u.a. aus der RestrictedUserInfo und der SwopCard Entität. Ersteres besteht aus den persönlichen Daten, die an andere User heraus gereicht werden dürfen, sobald ein Match entsteht. Damit dies möglich ist, mussten diese Daten in eine eigene Entität gekapselt werden. Die SwopCards Referenzen werden zudem direkt im User abgespeichert, damit schnell auf die eigenen Tauschkarten zugegriffen werden kann, ohne, dass eine Query-Anfrage gestellt werden muss.

Die Match Entität hält im Wesentlichen die Stati fest, ob ein User1 und/oder User2 den Match akzeptiert, abgelehnt oder (noch) nichts gemacht haben. Wir hatten zu Beginn den Status als eigene Entität modelliert, damit es als Enum fungieren und nur 3 Stati annehmen kann. Da wir die Match Objekte aber durch eigene Node-Module erzeugen, war das nicht mehr notwendig. Die weiteren Felder dienen dazu, zu bestimmen, welcher User welchen Status ändern darf. Sobald beide User akzeptiert haben, werden die RestrictedUserInfo - Objekten mit Referenzen befüllt und die ACL-Rechte der RestrictedUserInfo um Read-Rechte erweitert, sodass es den Usern nun möglich ist, die Kontaktdaten einzusehen.

Jeder User kann maximal 20 SwopCards erstellen. Sie halten die nötigen Informationen, damit ein Tausch stattfinden kann, also in welchem Kurs sich der User befindet und in welchen oder welche möchte er. Hierbei kam es zu derselben Einsicht, wie bei der Match Entität, sodass das Match-Feld ebenfalls einen String und keine eigene Entität hält.

Zudem gibt es noch die Course-Entität, die Daten der user-generierten Kurse hält. Dabei ist das validName ein Feld das angibt, ob der Kurs legitim ist oder nicht. Legitim bedeutet in dem Fall, ob der Kursname richtig geschrieben ist und auch die ID die korrekte ID ist. Dies wird aber erst durch später hinzukommende Admin-Funktionen relevant.

Match-Algorithmus

Wir matchen SwopCards nach bestimmten Kriterien: der User kann sich nie selber matchen, muss "im Pool sein" (der Status ist auf WAITING gesetzt), der eigene Kurse (und ggf. Gruppe) muss von der anderen SwopCard gesucht werden und andersherum, zudem haben ältere SwopCards eine höhere Priorität als jüngere. Passende SwopCards finden wir mittels durch Baqend bereitgestellten Query-Methoden.

Prozess der SwopCard-Erstellung

Bevor wir angefangen haben zu programmieren, haben wir uns klar aufgeschrieben, in welchen Zuständen sich SwopCards und Match-Objekte befinden können und welche Abhängigkeiten zwischen ihnen bestehen muss, wie z.B., dass sobald der Match von einem User abgelehnt wird die eine SwopCard wieder zurück "in den Pool" soll, wobei die andere den Status DECLINED annehmen soll. Die Funktionalitäten bzw. diese Prozesse sind in den Prozessdiagrammen [1] und [2] zu sehen (siehe Anhang).

Das erste Prozessdiagramm zeigt den Prozess an, wenn der User eine SwopCard erstellen will und das zweite Prozessdiagramm zeigt denjenigen Prozess an, wenn der User bereits eine SwopCard erstellt hat und ein Match gefunden wurde.

Schnittstelle

Da Vue das MVVM Pattern voraussetzt haben wir uns dafür entschieden ein eigenes Clientseitiges Model als Node-Modul zu schreiben, welches die gesamte Kommunikation mit der Datenbank steuert (siehe Anhang[12] S.21). Das ViewModel ist somit größtenteils losgelöst von der Kommunikation mit Baqend. Einzig allein die db.User.me Abfrage findet in den Vue Komponenten noch statt, die aber auch noch ausgelagert werden wird.

Das Model hält lokal ein Abbild des User-Objektes mit den dazugehörigen SwopCards. Das Abbild wird jedes mal aktualisiert, wenn der User eine Veränderung in der Datenbank bewirkt, z.B. mittels createSwopCard(), indem das aktualisierte User-Objekt als Response mitgesendet wird und dann in das lokale Abbild eingebunden wird. Damit ersparen wir uns jedes Mal die Datenbank abzufragen, nachdem eine Änderung vorgenommen wurde und senken damit die Anzahl der gesamten Requests. Die Userdaten werden als Feld im Model gespeichert.

Zu Beginn haben wir die Implementation über die Baqend Code Handler versucht, indem wir eigens geschriebene NodeJS Funktionen implementiert hatten. Da wir jedoch bereits wichtige ACLs zu den einzelnen Klassen gesetzt hatten und die Node-Rolle nicht über die Baqend Code Handler greift, hatten wir uns gegen die Handler entschieden und eigene NodeJS Module geschrieben. Zudem war es uns nicht möglich, weitere Parameter bei den Handlern zu übergeben, wie z.B. bei der Registrierung und der onValidate() Handler hatte nicht korrekt funktioniert, sodass wir mit Rücksprache des Baqend-Teams auf diesen Handler verzichten mussten.

Bei der Registrierung gibt es ein zusätzliches Feld "Display Name", welches jedoch im RestrictedUserInfo - Objekt gespeichert wird und nicht im User - Objekt. Da wir während eines von beiden Seiten akzeptierten Matches die E-Mail Adresse und den Display Namen für den jeweils anderen User freigeben, konnte man über die Handler nicht beide Objekte erstellen. Zudem wollten wir Redundanzen vermeiden, indem wir z.B. im User - Objekt ebenfalls den Display Namen speichern.

Ein weiterer Punkt gegen die Baqend Code Handler war, dass keine sog. "After" Handler ausgeführt werden konnte - die Handler greifen explizit nur während des Einfügen oder Manipulieren der Objekte ein, nicht aber nachdem man ein Objekt manipuliert hat.

Aus diesen Gründen haben wir uns entschieden die Schnittstelle zu Baqend hauptsächlich über selbst geschriebene Node-Module, sogenannte "Services", zu gestalten ([12]). In denen wir zusätzlich viel Business-Logik einarbeiten konnten. Einige Anfragen umgehen allerdings die Services, wie z.B. das Laden der Userdaten und der Kurse. Dadurch werden die Server entlastet, da dies mit Abstand die am häufigsten ausgeführten Anfragen sind. Die Erstellung

einer neuen SwopCard tritt im Gegensatz dazu nicht so häufig auf und läuft über unsere Services.

Kritik an unserer Schnittstelle

Im Nachhinein sind uns einige Schwachstellen unseres Entwurfes aufgefallen, wir haben uns aber im Rahmen des Praktikums und dessen zeitliche Beschränkung dafür entschieden bei unserer Modellierung mittels der Services zu bleiben.

Durch unsere Entscheidung die komplette Logik über Services zu steuern, ist es zur Zeit u.a. möglich Datenbank-inkonsistente SwopCards und Matches zu erstellen, d.h. z.B. eine leere SwopCard zu erstellen, indem man direkt die Datenbank anspricht (*new db.SwopCard().insert()*). Unter der Voraussetzung, dass man den Connection-String zu unserer Datenbank kennt. Das ist natürlich ein unerwünschtes und unsicheres Verhalten und müsste mittels Handler abgefangen werden. Rückblickend wären wir auch jetzt in der Lage die Services in die Handler so einzubetten, sodass eben jenes Verhalten nicht möglich wäre und die Stärken von Database und Backend as a Service komplett auszunutzen. Jedoch war uns zu Beginn nicht bewusst, sodass es zu solchen Komplikationen kommen würde, da wir noch nicht das nötige Wissen dafür hatten. In einer Release-fertigen Version würde das natürlich überarbeitet werden.

Fazit

Unser Anspruch an uns selbst war es, im Praktikum eine Anwendung zu entwickeln die eine reales Problem löst und damit Gebrauchswert schafft. Dass wir uns mit der Definition des minimal viable product bis zum Ende des Praktikums viel vorgenommen hatten war uns bewusst, gleichzeitig war auch unsere Bereitschaft, dafür etwas mehr Zeit als eigentlich für das Praktikum vorgesehen dafür zu investieren entsprechend hoch.

Anders als zu Beginn gedacht, haben wir uns relativ wenig in einzelnen Arbeitspaketen verrannt und mussten im späteren Verlauf wenige bereits implementierte Funktionen komplett über den Haufen werfen.

Durch die ständige Kommunikation im Team über Vergangenes und Anstehendes konnte jedes Teammitglied trotz der doch relativ klaren Aufgabenteilung einen guten Überblick über den Gesamtverlauf der Implementierung der Anwendung behalten.

Trotz der kurzen Praktikumsdauer von drei Wochen konnten wir wertvolle Metakompetenzen über die Umsetzung von und Arbeit mit einer Web-Application mit Datenbankbindung gewinnen. Darunter fällt zum einen das Wissen über möglicherweise auftretende Problematiken, aber auch die Sensibilität für den Aufwand vermeintlich kleiner Arbeitspakete sowohl in der Frontend-Programmierung als auch in der Backend-Programmierung.

Im Nachhinein hätten wir einige Dinge anders gemacht:

Zum Beispiel die stärkere Einbindung der Handler, wie auf Seite 14 beschrieben. Zudem die Erstellung von Dummy Daten, damit das Frontend-Team direkt damit in der GUI hätte arbeiten können. Dadurch hätte, dass das Frontend und Backend-Team noch gekapselter voneinander hätten arbeiten können.

Alle Präsentationsfolien können über <https://goo.gl/vYxKLB> eingesehen werden.

Anhang

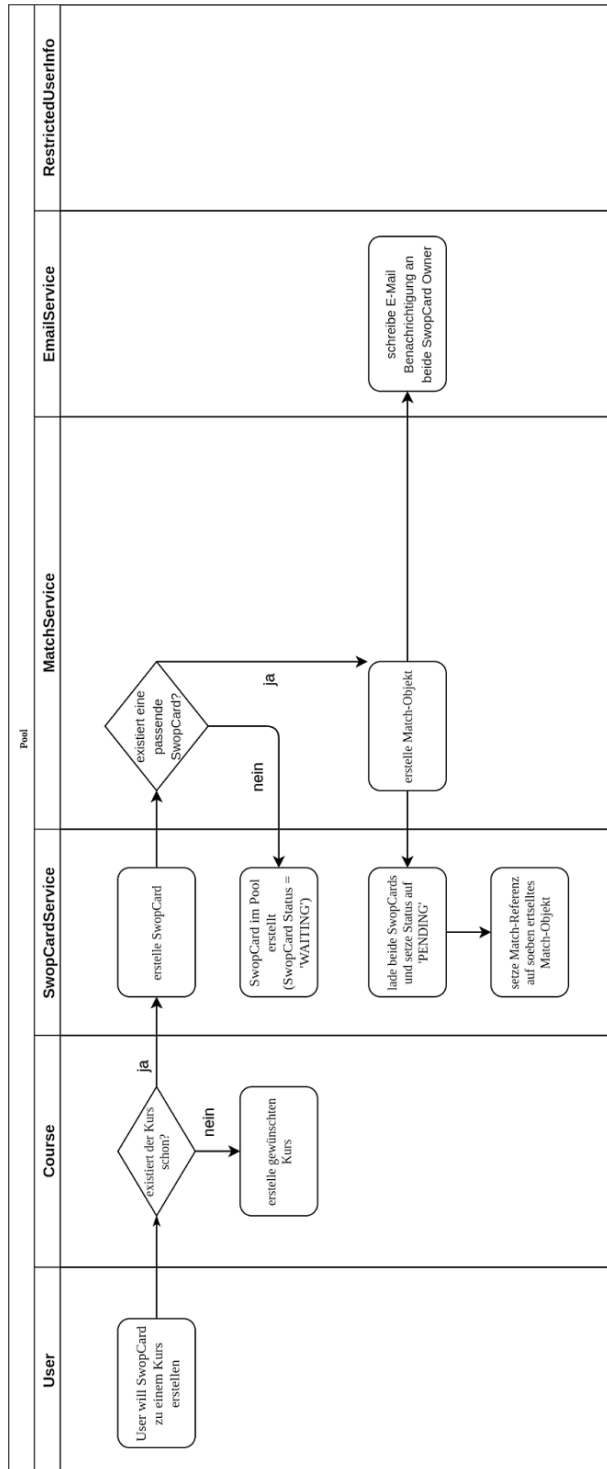


Abbildung 1. Prozessdiagramm, welches den Prozess einer SwopCard Erstellung darstellt [1]

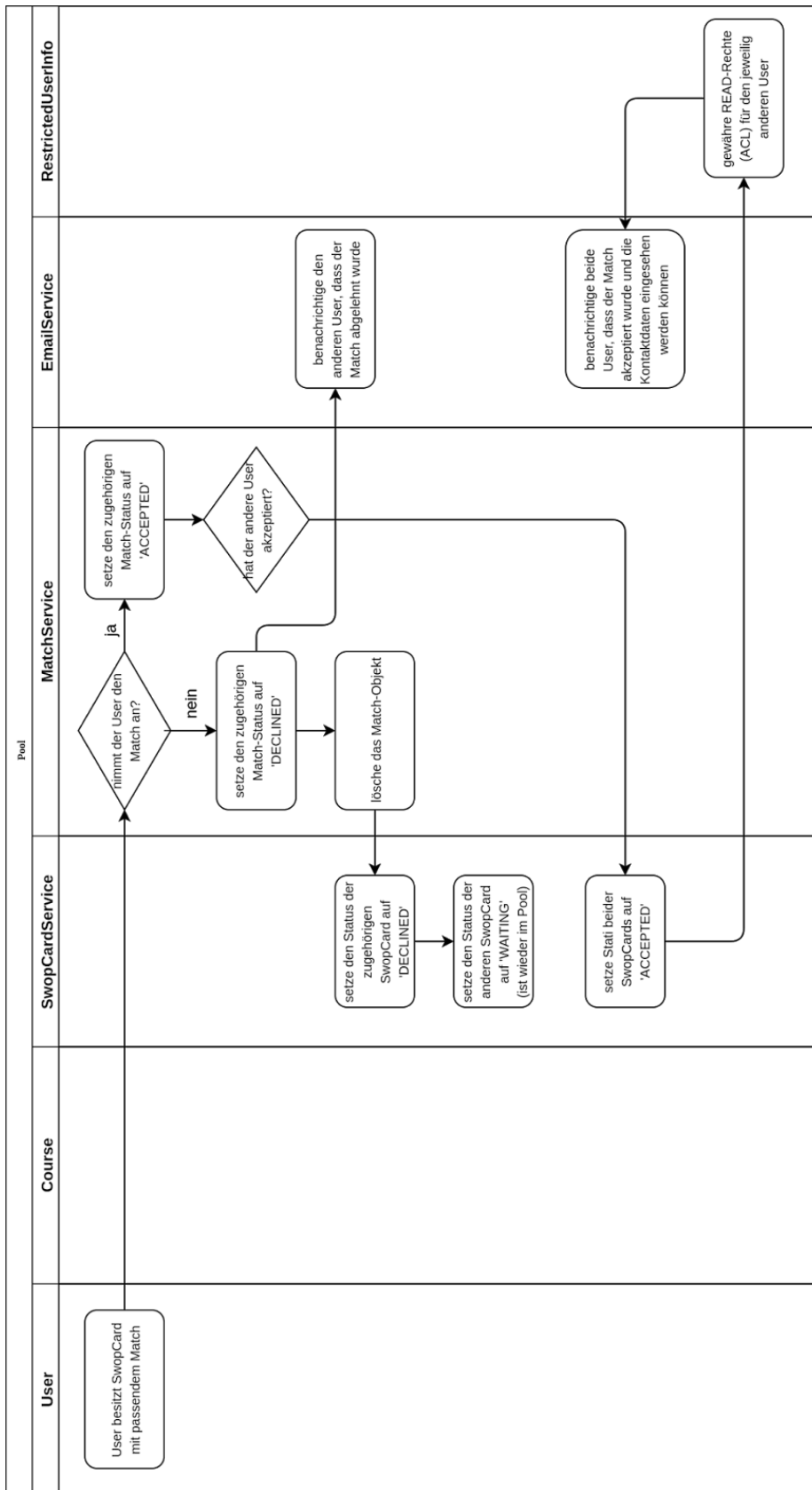



Abbildung 2. Prozessdiagramm, der den Prozess eines Matches von zwei SwopCards darstellt [2]




Swop ist das Tauschtool für Deine Uni-Veranstaltungen.

Login

Logge Dich mit Deiner Uni-Mail ein.

[Vergessen?](#)

[Noch kein Mitglied?](#)
[Einloggen](#)



Swop ist das Tauschtool für Deine Uni-Veranstaltungen.

Registrieren


Melde Dich mit Deiner Uni-Mail an und erstelle Deinen kostenlosen swop-Account.

☐ Ich habe die [Allgemeinen Gurkenbedingungen](#) gelesen und akzeptiere diese.


[Bereits Mitglied?](#)
[Registrieren](#)

Abbildung 3. Login-Screen [3]

Abbildung 4. Registrierungs-Screen [4]



Ausstehend
Alle
Match



Tausche 888-888
Hennings Herings Hexenfest
Gegen 333-333
Collins Cognac Content


24. August – 12:34 Uhr

[Löschen](#)

[Aktualisieren](#)

[+ Neue Tauschanfrage](#)

Abbildung 5. Dashboard-Screen [5]



Neue Tauschanfrage

Aus welcher Veranstaltung möchtest Du **heraus** wechseln?

- 111-111 – Annikas Ananas Anbetung
- 222-222 – Baltasars Bananenbrot Backkurs
- 1234 – Betrunknen Backen
- 333-333 – Collins Cognac Content
- 444-444 – Dagmars Damen Darkroom
- 555-555 – Evas Eventim Event

[Weiter](#)

Abbildung 6. Erste Seite des Tauschanfrage-Screens [6]

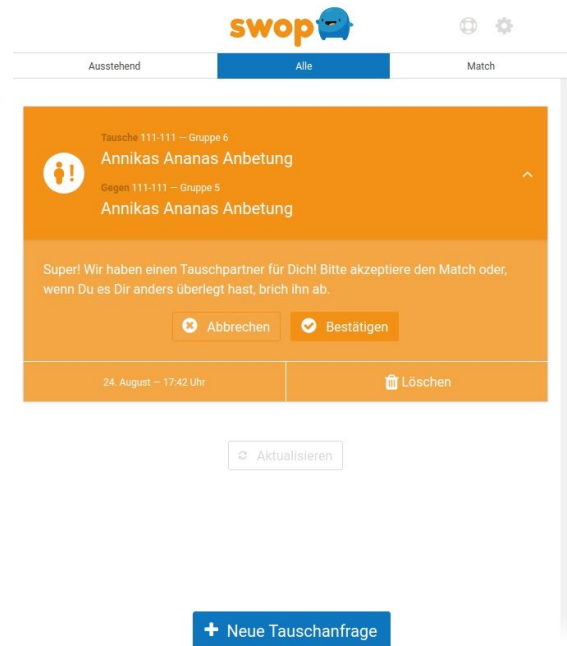
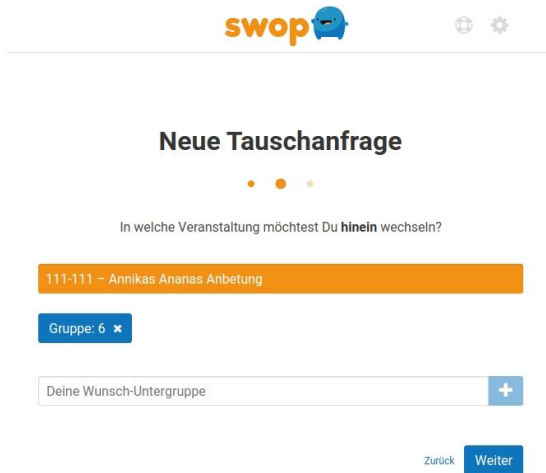


Abbildung 7. Zweite Seite des Tauschanfrage-Screens mit Gruppenbestätigung [7]

Abbildung 8. Dashboard-Screen mit vorhandenem Match [8]

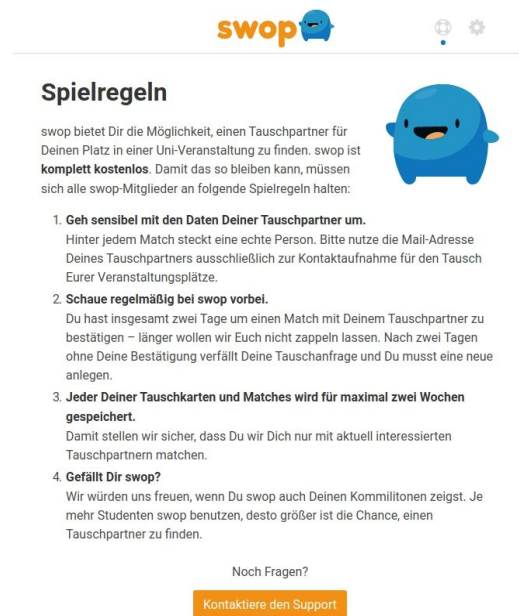
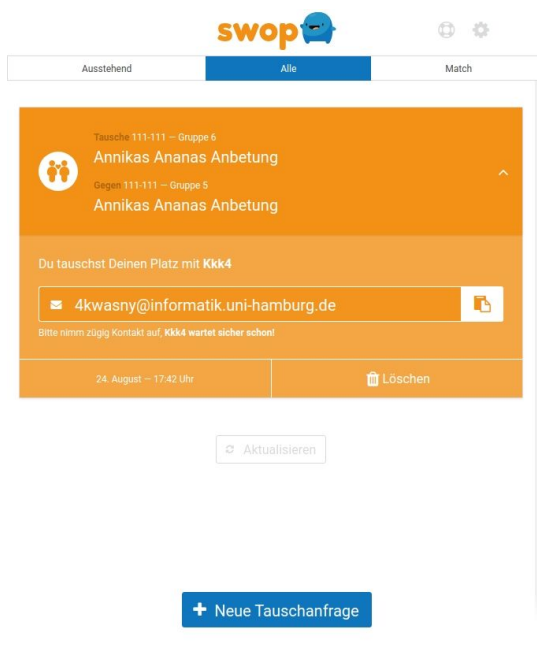





Abbildung 9. Dashboard-Screen mit vorhandenem und akzeptierten Match [9]

Abbildung 10. Spielregeln-Screen [10]


swop



Deine Einstellungen


Name ändern

Dieser Name wird Deinem Tauschpartner angezeigt.



Zusätzliche Mail-Adresse

Wenn Du eine zusätzliche Mail-Adresse hinterlegst, werden Benachrichtigungen aus swop nur noch an diese Adresse gesendet. Einloggen kannst Du Dich aber weiterhin **nur** mit Deiner verifizierten Uni-Mail (**4kwasny@informatik.uni-hamburg.de**).



Passwort ändern


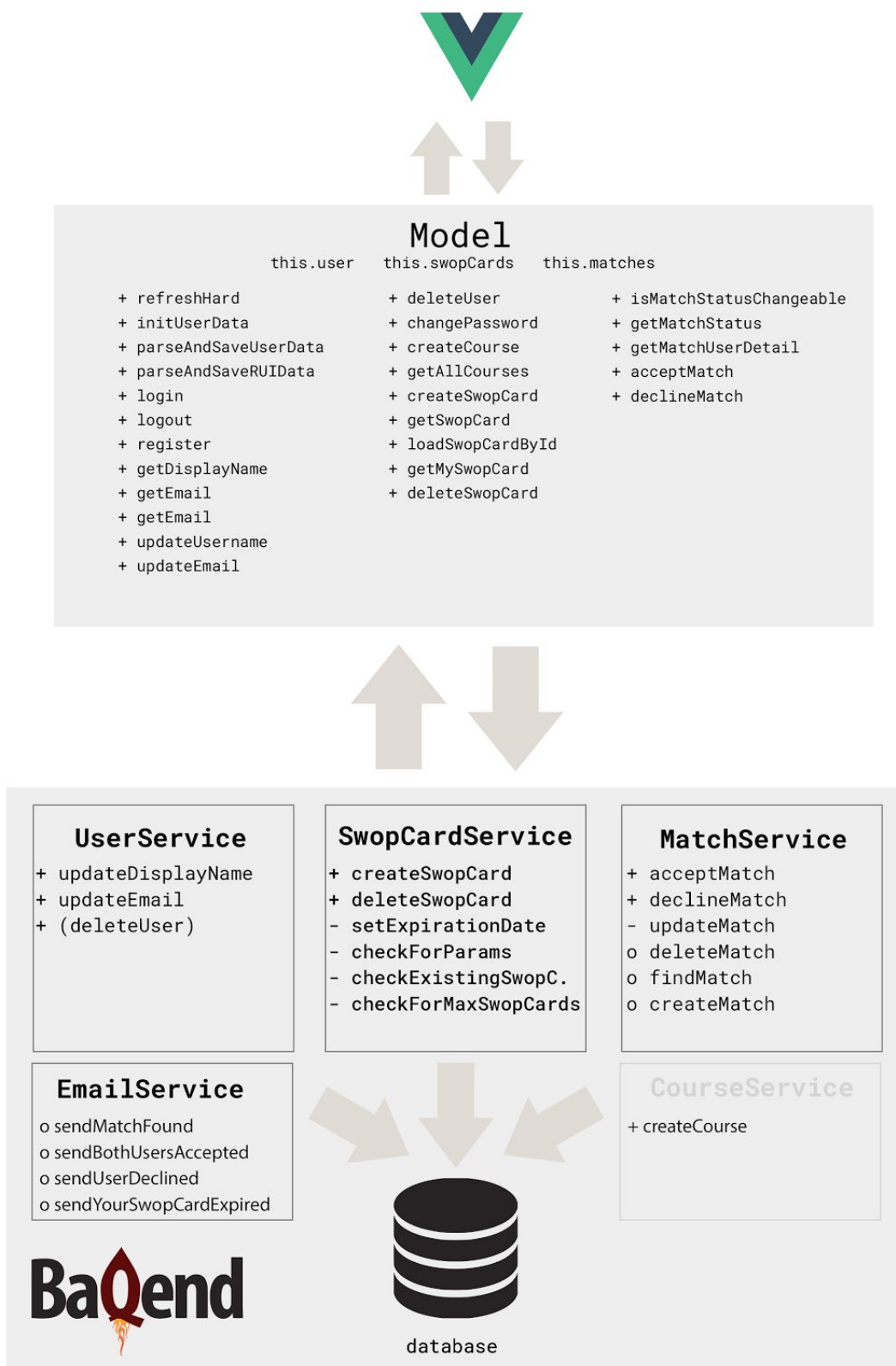
[Account löschen](#)

Abbildung 11. Profil-Screen [11]



Legende: "+" kann vom User/VieModel mittels Post aufgerufen werden, "o" kann von anderen NodeModulen aufgerufen werden, "-" Methoden die nur innerhalb des Moduls aufgerufen werden.

Abbildung 12. Schnittstelle Model/Baqend [12]