Fontys Hogeschool Techniek en Logistiek

Informatik

Software Factory

Work - to - Students

Implementierung

16. Januar 2018

Informationen

Autoren	Julia Mödinger (2593467)
	Moritz Richter(2620731)
	Thilo Ritzerfeld (2462966)
	Merve Sahin (2621363)
	Niklas Schwerin (2645386)
Modul	Software Factory
Betreuender Dozent	Ferd van Odenhoven
Institut	Fontys Hogeschool Techniek en Logistiek
Studiengang	Informatik
Studienjahr	2017/2018
Ort und Datum	Venlo, 16.Januar 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Struktur

Während der Implementation der Applikation haben wir verschiedene Techniken und Frameworks benutzt. Die beiden wichtigen Technologien waren Ionic und Firebase. Firebase ist unsere Cloud-Datenbank und Ionic unser Framework mit dem wir die hybride App erstellt haben.

Das Strukturdiagramm stellt den Aufbau des Angular-Frameworks dar. Jede Ansicht wird in Form einer Klasse implementiert und als *Component* bezeichnet. *Component*s werden jeweils mit einer *Directive* versehen, die *Metadata* über das *Template* (HTML und CSS Datei) enthält. Das *Component* und *Template* werden über *Property Bindings* verknüpft, um das Befüllen von HTML Elementen zu ermöglichen. Mit *Event Binding* kann man events bzw. Methoden aus dem HTML-Template aufrufen. Dependency Injection erfolgt über *Injector*s. Diese stellen in der Regel tools oder primäre Funktionalitäten zur Verfügung, wie z.B. *NavController*.

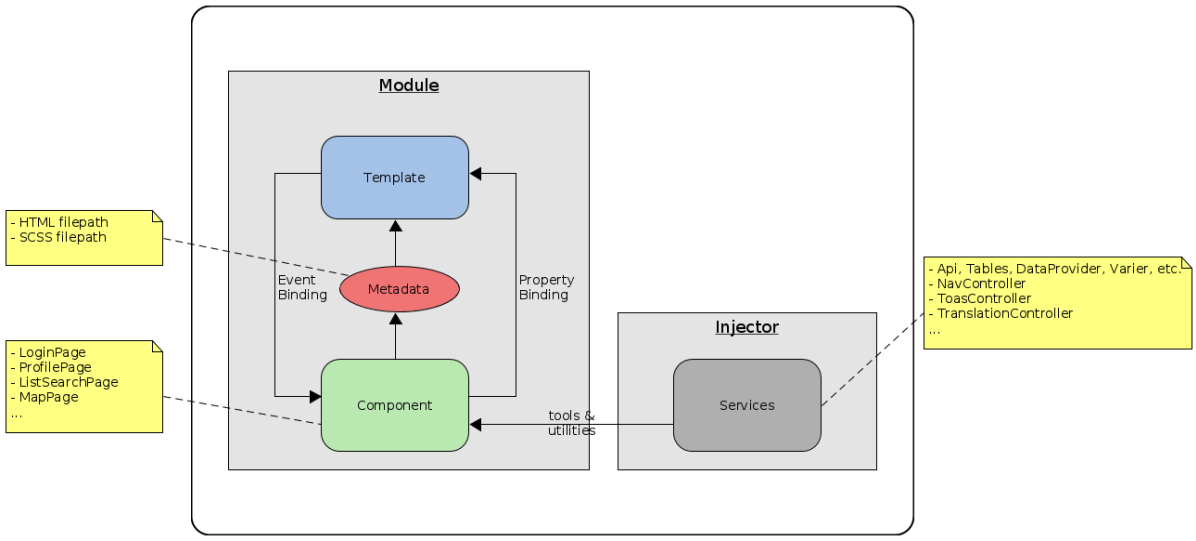

Abbildung 1: Strukturdiagramm

2 Sprachen

Eine der wesentlichen Anforderungen des Kunden war es, dass die App so aufgebaut wird, dass sie in zwei Sprachen (Deutsch & Englisch) verfügbar ist. Für die Zukunft sollen aber die Grundsteine gelegt werden, um einfach weitere Sprachen zu implementieren. Hierbei ist zu sagen, dass ausschließlich die Navigation der App selbst in verschiedenen Sprachen verfügbar sein wird, für die Inhalte, welche vom Nutzer eingetragen werden, ist der Nutzer selbst verantwortlich.

Um die Benutzerfreundlichkeit zu erhöhen liest die App die Spracheinstellungen des Endgerätes aus, so dass bereits beim Öffnen alle Eingabehilfen in der Sprache des Nutzers angezeigt werden.
Da in dieser Version der App ausschließlich Deutsch und Englisch implementiert werden, wird für anderssprachige Nutzer die App in Englisch angezeigt, da Englisch die Weltsprache ist.

```
1   var deviceLang;
2   global.getPreferredLanguage().then(result => {
3     deviceLang = result.value;
4     switch (deviceLang) {
5       case 'de-DE':
6         translate.use('de')
7         break;
8       case 'de':
9       translate.use('de')
10      break;
11      case 'Deutsch':
12      translate.use('de')
13      break;
14      case 'en-US':
15        translate.use('en');
16        break;
17      case 'en-GB':
18        translate.use('en');
19        break;
20      case 'en-IN':
21        translate.use('en');
22        break;
23      case 'en-AU':
```

```
24            translate.use('en');
25            break;
26          case 'en-US':
27            translate.use('en');
28            break;
29          case 'en':
30          translate.use('en')
31          break;
32          default:
33            translate.use('en');
34        }
35      });
```

Damit die App in den verschiedenen Sprachen, auch alle Texte und Wörter in der jeweiligen Sprache anzeigt, haben wir ngx-translate benutzt. Das ist eine Libary für Angular, welche es ermöglicht Übersetzungen in verschiedenen Sprachen zu definieren. Zwischen den einzelnen Sprachen kann während Benutzung der App gewechselt werden. Für jede Sprache muss eine Art Wörterbuch angelegt werden, indem die Übersetzung hinterlegt wird. In dem Code der App wird dann nur ein zugehöriger Schlüssel genannt. Dieser Schlüssel wird in dem Wörterbuch gesucht und in der App wird die Übersetzung angezeigt. Wenn in dem Wörterbuch kein Eintrag zu diesem Schlüssel steht, wird der Schlüssel in der Sprache der App angezeigt.

```
1 <button #myButton *ngIf="!isOwn" id="message"
2     ion-button sendMessage color="wts_grey"
3         (click)="getConversation()">
4   <ion-icon id="message" name="msg">
5       {{ 'SENDMESSAGE' | translate }}
6     </ion-icon>
7 </button>
```

Für die Zukunft der App besteht die Möglichkeit über einen Translation-Service auch die Nutzereingaben in die entsprechende Sprache zu übersetzen. Beispielsweise ist es akutell für einen englischsprachigen Arbeitgeber nicht möglich das Profil eines deutschen Studenten zu verstehen, wenn der Student seine persönlichen Eingaben in Deutsch gemacht hat. Aus diesem Grund wird der Arbeitgeber sich mit hoher Wahrscheinlichkeit nicht mit dem Studenten in Verbindung setzen, obwohl der Student womöglich die perfekten Voraussetzungen und Interesse für die Arbeit mitbringt.

Dies stellt eine große Barriere zwischen verschiedensprachigen Profilen/Personen dar. Wenn zukünftig auch die persönlichen Eingaben in die Sprache des jeweiligen App-Nutzers übersetzt werden könnten, wäre dies ein großer Vorteil um internationale Kontakte zu knüpfen und sich global zu vernetzen.

Die drei größten Anbieter für diese Art der Übersetzung sind Google, Microsoft und deepL, wobei bei die durch kommerzielle Nutzung der Übersetzung in jedem Falle Kosten entstehen würden.
Ein Vorteil der API von Google ist, dass sie die meisten Sprachen (104) beinhaltet, nachteilig ist, dass ein pauschaler Preis von von 20 Dollar pro 1 Mio. Zeichen zu zahlen ist.[1]

Microsoft hingegen kostet mit 8,43 € pro 1 Mio. Zeichen im niedrigsten Tarif am wenigsten und bei starkter Nutzung der App, und somit Aufsteigen im Tarif, der Preis verhältnismäßig geringer.[2] Allerdings bietet Microsoft im Gegensatz zu Google auch ausschließlich 60 verschiedene Sprachen an.[3]

Mit Abstand die wenigsten Sprachen bietet der Übersetzer deepL an, wobei hier momentan nur sieben Sprachen angeboten werden. Diese sind Englisch, Deutsch, Französisch, Spanisch, Niederländisch, Polnisch und Italienisch, jedoch folgen bald weitere Sprachen, wie Mandarin, Japanisch und Russisch.[4] **Ein weiterer Nachteil ist, dass die API noch sehr neu und damit nicht sehr erbrobt ist. Angaben zu den exakten Kosten können ebenfalls noch keine gemacht werden.** Dennoch hat deepL einen entscheidenen Vorteil, und zwar hat DeepL in Blind Test im Vergleich mit Microsoft und Google außerordentlich gut abgeschnitten und übertrifft die Konkurrenz um ein vielfaches in der Qualität der Übersetzung in jeglicher Sprachkombination.[5]

Somit besteht hier die Empfehlung zukünftig eine Übersetzung der Nutzereingaben einzuführen um damit die Kommunikationsbarriere zu entfernen. Trotz der wenigen Sprachen und aufgrund der hohen Qualität der Übersetzungen ist die Schnittstelle von deepL als zu empfehlen. Danach

[1] [https://cloud.google.com/translate/?hl=de (12.01.2018)]
[2] [https://azure.microsoft.com/de-de/pricing/details/cognitive-services/translator-text-api/ (12.01.2018)]
[3] [https://translator.microsoft.com/de/help/articles/languages/ (12.01.2018)]
[4] [https://www.deepl.com/api-reference.html (12.01.2018)]
[5] [https://www.deepl.com/press.html (12.01.2018)]

folgt der Übersetzer von Microsoft, da der Preis sich dauf die Nutzung der App anpassen lässt und die wichtigsten Sprachen enthalten sind. Zuletzt folgt die API von Google, obwohl hier zwar die meisten Sprachen enthalten sind, ist der Preis pauschal und nicht auf die Nutzung der App anpassbar.

3 Sprints

Die Implementierung der mobilen Applikation Work-to-Students (WTS) findet nach dem agilen Vorgehensmodell Scrum statt. Innerhalb von 8 Wochen findet die schrittweise Implementierung der Applikation statt. Scrum ermöglicht eine sequentielle Fokussierung auf einzelnen Use Cases, so dass schon nach kurzer Zeit ein Teil der Software voll funktionsfähig ist.

3.1 Sprint 1 - Initialisierung des Projektes & Einarbeitung

In Sprint 1 findet die Initialisierung des Projektes und die Einarbeitung in die neue Arbeitsumgebung statt. Die Entscheidung nicht sofort mit der Implementierung der ersten Funktionalitäten zu beginnen wurde aktiv getroffen, um Fehler zu vermeiden und jedem Gruppenmitglied Zeit zu geben um sich zurecht zu finden, so dass sich niemand mit der Implementierung einer Funktionalität überfordert fühlt. Zu Beginn der Einarbeitungszeit wurde mit einer Beispiel-App gearbeitet, welche Ionic bereitstellt, um das Prinzip der hybriden App-Entwicklung und das in einander Greifen von HTML-, SCSS- und Typescript-Dateien zu verstehen. Des Weiteren wurde bereits geschaut welche Teile der Beispiel-App auf die Work-to-Students App übertragen werden können.

3.2 Sprint 2 - Login, Logout & Einarbeitung

Sprint 2 enthält weiterhin Einarbeitungzeit, da nun auch in der Implementierung Fehler auftreten können und erlernt werden muss wie diese zu verstehen sind. Bei der Implementierung des Logins wurde die Entscheidung getroffen den Verschlüsselungsalgorithmus SHA256 zu verwenden, da dieser einer der zur Zeit sichersten Verschlüsselungsmethoden ist. Zuerst wurde eine schlichte einfache Oberfläche erstellt. Diese hat neben den zwei Textboxen für E-mail und Password und dem Knopf zum Einloggen auch noch einen Link, welcher auf die Webversion von WorktoStudents verweisen soll um die Erstellung eines Benutzerkontos durchzuführen, wenn noch keins erstellt wurde. Da die Webversion allerdings noch nicht öffentlich zugänglich ist, wird momentan noch auf www.google.com weitergeleitet.
Im Backend wird das Passwort mit SHA256 verschlüsselt und zusammen mit der E-Mail zur Datenbank geschickt und verglichen. Dank dieser vorgehensweise steht kein Passwort im Klartext in der Datenbank. Dadurch ist sicher gestellt, dass selbst wenn jmd die Datenbank ausliest er keine Möglichkeit hat die Passwörter zu lesen. Wenn der vergleich von den verschlüsselten

Passwortern erfolgreich war, wird auf das persönliche Profil des Benutzers weitergeleitet.
Der Logout ist eine einfache Weiterleitung auf die LogIn-Seite, von der keine anderen Seiten erreicht werden können, ohne dass man sich angemeldet hat.

3.3 Sprint 3 - Profilanzeige & Profilbearbeitung

In Sprint 3 wurde mit der Implementierung eines Studentenprofiles begonnen. Nachdem die Daten im Profil korrekt ausgelesen wurden, wurde die vereinfachte Profilbearbeitung implementiert. Da die mobile Version nur eine schnelle Option für unterwegs sein soll haben wir uns dazu entschieden ausschließlich die wichtigsten Merkmale zur Karriere des Studenten (Geburtstag, Universität, Studiengang, Abschluss, Studienfortschritt, voraussichtliches Studienende, Beschreibung & Beschäftigung) bearbeiten zu können. Anschließend wurden auch die Profile von Universität und Arbeitgeber implementiert. Diese beiden Nutzergruppen haben keine Möglichkeit die Profile in der mobilen Version zu modifizieren, da die Profile hauptsächlich zur Information der Studenten dienen und Work-to-Students von Arbeitgebern primär über die Web-Version genutzt werden wird. Universitäten haben keinen Zugang zur mobilen Applikation und können ausschließlich die Web-Version nutzen. Es wurde eine weitere Klasse implementiert, welche dazu dient um den Nutzer auch bei der späteren Suchfunktion auf das richtige Profil navigieren zu können. Hier wird überprüft aus welcher Quelle heraus navigiert wird (z.B. Menübar), ob eine Account-Id übergeben wird und ob diese zu dem eigenen oder zu einem externen Profil gehört. Je nach Fall wird dann zu dem enstprechenden Profil navigiert.

3.3.1 Profilanzeige

Um die Profile der bestimmten Usergruppen anzeigen zu können, müssen wir die Daten aus den richtigen Feldern der Datenbank lesen und in die erstellten Felder eintragen. Hierbei wird unterschieden, ob es sich um das eigene Profil handelt, oder ob man sich ein Profil einer anderen Nutzergruppe - Unviersität oder Unternehmen- anzuschauen. Dafür erstellen wir mithilfe der Funktionen „getByValue“, „filterByValue“ und „getById“ die zugehörigen Datenbankabfragen, die uns dann die zu den einzelnen Usergruppen und dem zugehörigen Profil hinterlegten Informationen anzeigt.

```
1   load() {
2    if (this.isOwn == false) {
3      console.log("Profile is extern, printed in profile.ts");
```

```
      this.accID = this.navParams.get("userId");
      this.StudentTable.getByValue("Account_Id", this.accID,
   "student-abfrage", this.onComplete);
      this.AccountTable.getById(this.accID, "account-abfrage",
    this.onComplete);
      this.StudentPassionTable.filterByValue("Account_Id", this.accID,
   "passionStudent-abfrage", this.onComplete);
      this.StudentSkillTable.filterByValue("Account_Id", this.accID,
    "skill-abfrage", this.onComplete);
    } else {
      console.log("Profile is own, printed in profile.ts");
      this.StudentTable.getByValue("Account_Id", this.accID,
 "student-abfrage", this.onComplete);
      this.AccountTable.getById(this.accID, "account-abfrage",
this.onComplete);
      this.StudentPassionTable.filterByValue("Account_Id", this.accID,
"passionStudent-abfrage", this.onComplete);
      this.StudentSkillTable.filterByValue("Account_Id", this.accID,
"skill-abfrage", this.onComplete);
    }
  }
```

Damit man foglerichtig auch auf das richtige Profil weitergeleitet wird, dass man sich anschauen mag, haben wir in unserem „Varier“eine zugehörige Funktion implementiert. Dies wird mit einer „ID “überprüft.

```
load(id) {
        //Checks if navigation was made from side menu.
        //Navigates to own profile
        if (this.userId == undefined) {
            console.log("Profile_Extern: Own profile reached from Menubar
               ");
            this.isOwn = true;
            this.accID = id;
            this.hasContact = true;
            this.AccountTable.getById(this.accID, "account-abfrage", this
               .onComplete);

            // Navigates to own profile
        } else if (id == this.userId) {
```

```
13              console.log("Profile_Extern: Own profile reached from Login")
                    ;
14              this.isOwn = true;
15              this.accID = id;
16              this.hasContact = true;
17              this.AccountTable.getById(this.accID, "account-abfrage", this
                    .onComplete);
18
19              //Navigates to foreign profile
20          } else {
21              console.log("Profile_Extern: Extern profile reached");
22              this.isOwn = false;
23              this.accID = this.userId;
24              this.hasContact = false;
25              console.log("accID ist jetzt: " + this.accID);
26              this.AccountTable.getById(this.accID, "account-abfrage", this
                    .onComplete);
27          }
28      }
```

3.3.2 Profilbearbeitung

Um dem Studenten die Möglichkeit zu geben ihre eigenen Profildaten zu ändern, wurde die Klasse "ProfileEditPageërstellt. Ebenso, wie in der Profilanzeige, werden die gespeicherten Daten aus der Datenbank geladen, sodass Sie für den Nutzer in input-Feldern dargestellt werden können. So weiß der Nutzer, welche Daten momentan in seinem Profil gespeichtert sind und kann diese ändern in dem er eine neue Eingabe in den input-Feldern macht. Um die Änderungen zu speichern bestätigt der Nutzer seine Eingabe durch einen Button-Klick und die Methode save() wird ausgeführt. Durch save() wird keine Überprüfung gemacht, in welchen Feldern sich Werte geändert haben, daher werden die Werte aller Felder in der Datenbank überschrieben. Des Weiteren navigiert diese Methode den Nutzer zu seinem eigenen Profil zurück.

```
1       save() {
2         this.studentjson.body.Uni = this.Uni;
3         this.studentjson.body.Abschluss = this.Abschluss;
4         var date = moment(this.Abschluss_Datum,
5       "YYYY-MM-DD").format("DD.MM.YYYY");
6         this.studentjson.body.Abschluss_Datum = date;
```

```
7       var birthdate = moment(this.Geb_Datum,
8     "YYYY-MM-DD").format("DD.MM.YYYY");
9       this.studentjson.body.Geb_Datum = birthdate;
10      this.studentjson.body.Studiengang = this.Studiengang;
11      this.studentjson.body.Semester = this.Semester;
12      this.studentjson.body.Nachname = this.Nachname;
13      this.studentjson.body.Name = this.Name;
14      this.studentjson.body.Beschaftigung = this.Beschaftigung;
15
16    this.StudentTable.update(this.studentjson.id,
17  this.studentjson.body, "", function (flag, json) {
18      if(json){
19
20        this.translate.get("SAVED_CHANGES").subscribe( value =>{
21          const toast = this.toastCtrl.create({
22            message: value.SAVED_CHANGES,
23            duration: 3000,
24            position: 'top'
25          });
26          toast.present();
27        });
28      }
29    });
30    this.navCtrl.popToRoot();
31    this.navCtrl.setRoot(StudentProfilePage, { userId:
32  this.accID, isOwn: true, hasContact: true });
33
34  }
```

Falls der Nutzer seine eingetragenen Werte nicht speichern möchte hat er die Möglichkeit dies durch Button-Klick zu bestätigen. In diesem Fall werden keine Werte in der Datenbank überschrieben und der Nutzer wird ebenfalls wieder zu seiner eigenen Profilseite navigiert.

3.4 Sprint 4 - Listensuche & Kartensuche

Sprint 4 beinhaltet die Implementierung der Listensuche bzw. die Einbindung der externen Kartensuche.

3.4.1 Listensuche

Bei der Listensuche sind einige Probleme aufgetreten. Auf der einen Seite war die Frage, wenn WorkTostudents später sehr viele Benutzter hat, wie schaffe wir es das die Suche mit diesen Datenmengen trotzdem noch performant läuft. Auf der anderen Seite, mussten wir sicherstellen, dass das Endgerät auch in der Lage ist die Daten anzuzeigen.
Diese Problem haben wir behoben, indem wir die Suche Serverseitig durchführen und nur eine bestimmte Anzahl an Treffern anzeigen. Wir zeigen erstmal nur X Treffer in der Liste an und sobald der Benutzer die Liste durchggescrollt hat, werden Y weitere Eintrage geladen. Dieses vorgehen wiederholt sich so lange bis der Server keine neuen Einträge zur verfügung stellen kann.
Die Suche funktioniert über verschiedene Suchparameter. Das beduetet, dass in der Suche nach einem bestimmten Merkmal gesucht wird. Beispielsweise wird nach allen Studenten gesucht, die Max heißen. Dazu wird zuvor der Name als Suchparameter ausgewäht und anschließend Max in die Suchleiste eingtragen. Sobald die Suche 0.5 Sekunden nicht mehr verändert wurde und keine Suche momentan läuft wird eine neue Suche angestoßen.

```
1  searchForStudents() {
2      if (this.searchParameterStudent.length > 0 &&
3      this.searchParameterStudent != "Name") {
4        this.StudentTable.getAllContaining(this.searchParameterStudent,
5        this.filter, "student-search-query", this.onComplete);
6      } else if (this.searchParameterStudent.length > 0) {
7        if (this.filter.indexOf(" ") !== -1) {
8          var paras = this.filter.split(" ");
9          paras.forEach(element => {
10           this.StudentTable.getAllContaining("Name", this.filter,
11           "first-name-search-query", this.onComplete);
12         });
13       } else {
14         this.StudentTable.getAllContaining("Name", this.filter,
15         "first-name-search-query", this.onComplete);
16       }
17     }}
```

Dieser Codeabbschnitt zeigt die Methode searchForStudents(), welche vom infinityscroll angestoßen wird, sobald das Ende der Liste erreicht ist. Zuerst wird überprüft ob es Suchparameter gibt und dieser nicht "Nameïst. Dann kann die Suche nämlich ganz normal auf der Datenbank laufen. Falls der Suchparameter allerdings "Nameïst wird zuvor auf Freizeichen geprüft und anschließend der Suchparameter in meherere einzelne aufgeteilt. dies führt dazu, dass man sowahl nach "Max Mustermannäls auch nach "Mustermann Maxßuchen kann und die selben Suchergebnisse angezeigt werden.

3.4.2 Kartensuche

Für die Kartensuche hat unser Kunde „Screenware“ uns zum Ende des Sprints den Link zu einer Test-Karte, welche nicht Live ist, bereitgestellt. Damit wir mit dieser Karte arbeiten konnten, erhielten wir ebenso zwei Schnittstellenbeschreibungen: zum Anlegen eines Placemarks und zur Bearbeitung dessen.
Da es sich hierbei um eine Funktion eines externen Dienstleister handelt, haben wir uns entschieden nicht direkt mit der Implementierung zu beginnen, sondern erst einmal uns mit der bereitgestellten Karte vertraut zu machen und die zugehörigen Schnittstellenbeschreibungen zu verstehen. Nachdem wir relativ zügig Placemarks auf der Karte setzen konnten -leider nur über das Backend Management System- stießen wir auf Probleme hinsichtlich der Implementierung, damit mithilfe der Applikation ein Pin auf dieser Karte gesetzt werden konnte. Hierzu werden verschiedene Parameter, wie z.B. die ID der Karte, oder ein Passwort, das einem Zugang zur Karte gibt, oder einer backwardURL, die nach Erstellen des Pins aufgerufen wird, damit wir eine ID des Pins zurückerhalten, um diesen später bearbeiten zu können, verwendet. Nachdem wir feststellten, dass wir mithilfe der Beschreibungen nicht vorwärts kamen, hielten wir noch einmal Rücksprache mit unserem Kunden. Infolgedessen erhielten wir eine URL, die wir verwenden können, damit wir einen Pin erzeugen können und die ID dessen zurückgegeben bekommen. Im Anschluss daran konnte nun die eigentliche Implementierung für das Erzeugen eines Pins begonnen werden.

Implementierung der Kartensuche

```
1 setPin(): any {
2     var url = "http://www.ynfynyty.net/mapapp/api/create.aspx
3     mapid=509&pwc=hg67UH!&backwardurl=www.testing.worktostudent.com";
4
```

```
5     var sid = "";
6     var info = this.httpClient.get(url);
7     return new Promise((resolve)=> {
8       info
9       .subscribe(data => {
10      }, error => {
11        var index = error.url.indexOf("sid=") + 4;
12        sid = error.url.substr(index);
13        resolve(sid);
14      });
15    })
16  }
```

Damit wir einen Pin auf der Karten erzeugen können, müssen wir erst einmal eine URL erzeugen. Diese URL besteht aus dem Link zur Karte, der zugehörigen mapid, dem Passwort zu dieser Karte und einer backwardurl. Diese einzelnen Parameter sind notwendig, damit die Pins alle auf derselben Karte erzeugt werden können. Die backwardurl ist in unserem Fall ebenfalls wichtig, denn mithilfe dieser erhalten wir im Anschluss an das Erzeugen des Pins eine URL, die eine ID beinhaltet, nämlich die sid, die jeden einzelnen Pin identifiziert. Nachdem es nun möglich ist, einen Pin zu erzeugen, wollten wir die zugehörige sid zurückgegeben bekommen, damit wir das editieren der einzelnen Pins implementieren konnten. Doch leider stießen wir dort auf das nächste Problem, denn die sid wird in unserem Fall leider nur in einer Fehlermeldung zurückgegeben. Dies hat damit zu tun, dass wir nach ausführen der Funktion einen Pin zu setzen, normalerweise auf eine Seite weitergeleitet werden, damit wir die sid zum Editieren des Pins verwenden können. Doch leider fuktioniert das Weiterleiten, wie es im Browser auf einem Computer funktionieren sollte, nicht wie gewünscht in der Applikation, da es Problemen mit dem „Cross-Origin Resource Sharing“ gibt. Natürlich können wir die sid uns aus der Fehlermeldung herausnehmen, doch findet ein anderer Fehler statt, kann die sid nicht entnommen werden und somit der Pin nicht bearbeitet werden. In Rücksprache mit unserem Kunden wurde nur die Möglichkeit einen Pin zu setzen implementiert.

3.5 Sprint 5 - Nachrichtenverwaltung & Kontaktanfrage

Dieser Sprint befasst sich mit der Nachrichtenverwaltung und Kontaktanfrage.

3.5.1 Nachrichtenverwaltung

Die Nachrichtenverwaltung setzt sich aus zwei Ansichten zusammen: Nachrichtenliste und Konversation. Diese Funktionalitäten stellen die Möglichkeit dar, zwischen anderen WTS-Nutzern persönlichen Kontakt aufzubauen.

Nachrichtenliste

Dem Kunden zufolge soll die Nachrichtenliste bestimmte Merkmale aufweisen: die Liste soll nach Aktualität der Konversationen sortiert werden, sodass aktuelle Nachrichten ganz oben zu sehen sind, ungelesene Nachrichten sollen markiert und jeweils mit einem Zeitstempel versehen werden.

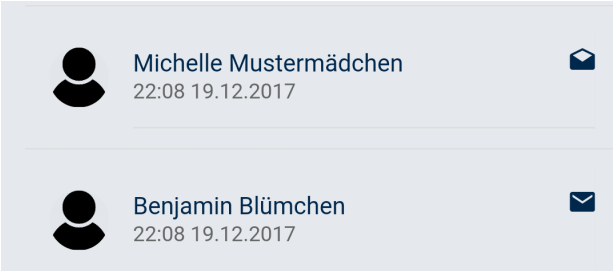

Abbildung 2: Nachrichtenanzeige

Um die Liste mit Daten zu befüllen wurde eine *load()*-Methode implementiert, die einen *Promise* zurückgibt:

```
1 load(): Promise<void> {
2     var promise: Promise<void> = new Promise<void>((resolve, reject) => {
3       this.messageArray = new Array();
4       this.users = new Array();
5       this.storage.get("user_id").then((id) => {
6         this.accID = id;
7         this.conversationTable.filterByValue("Account_Id_1", id, "query",
              this.onComplete);
8         resolve();
```

```
9         });
10      });
11      return promise;
12  }
```

Nach dem Initialisieren der *Arrays* wird die *ID* des Nutzers aus der lokalen Datenbank *(Storage)* geholt, um über diesen die Konversation aus der Datenbank abzufragen. Sobald die Abfrage ausgeführt wurde, werden die Ergebnisse der *this.onComplete*-Methode übergeben

Konversation

Beim Implementieren der Konversation musste darauf geachtet werden, dass Nachrichten in richtiger Reihenfolge geholt wurden. Da Firebase nicht die Möglichkeit bietet Einträge nach bestimmten Parametern zu sortieren, musste serverseitig eine Funktion implementiert werden, die diese Funktionalität ersetzt.

```
1  admin.initializeApp({
2  credential: admin.credential.applicationDefault(),
3  databaseURL: "https://worktostudents.firebaseio.com"
4  });
5
6  exports.sortBy = functions.https.onRequest((req, res) => {
7    cors(req, res, () => {
8    var results = [];
9    var sorted = [];
10
11   var ref = admin.database().ref().child(req.body.tbl);
12   var query = ref.orderByChild(req.body.key).equalTo(req.body.value);
13
14   if(req.body.limit > 0){
15     query.limitToFirst(req.body.limit);
16   }
17
18   query.once('value').then(snapshot => {
19     snapshot.forEach((json) => {
20     results.push({
21     id:json.key,
22     body: json.val()
23     });
```

```
24      })
25      }).then(() =>{
26        function mergeSort(arr){
27          ...
28        }
29        sorted = mergeSort(results);
30        res.status(200).send(sorted);
31      });
32    });
33  });
```

Der Name der Funktion wir in Zeile 6 mit *sortBy* definiert. Aus diesem wird später eine URL generiert, mit der man diese Funktion aufrufen kann. Da die *sortBy*-Funktion Parameter erwartet werden diese über das *body* einer *HTTP-POST* übersendet. Auf das *body* der POST Methode kann man mit *req.body* (Zeile 11) zugreifen. Als erstes wird die Datenbank-Schnittstelle mit *admin.initializeApp* initalisiert und über die Parameter des *body*s abgefragt. Die Daten werde zunächst in der *results* Array abgespeichert und dann mit der *mergeSort*-Funktion nach dem Zeitstempel sortiert und zurückgegeben.

Klientseitig wird zunächst in der *ngAfterViewInit*-Methode die Datenbank abgefragt:

```
1  this.messageTable.getByKeyValueSortedBy("Konversation_Id", this.id, "
       Zeitstempel", "nachrichten-abfrage", this.onComplete, 0, true, 15);
```

3.5.2 Kontaktanfrage

Die Kontaktanfrage soll verschiedene Funktionen erfüllen. Der Nutzer soll eine Kontaktanfrage verschicken und gesendet können, und soll diese dann akzeptieren oder ablehnen können. Weiterhin kann der Nutzer der Kontaktanfrage eine Nachricht mitsenden. Sendet ein Unternehmen einem Studenten eine Anfrage, ist die Nachricht ein Pflichtfeld. Sobald der Nutzer die Kontaktanfrage bestätigt soll der andere Nutzer in seinem Netzwerk auftauchen wie im folgenden Kapitel "Netzwerkübersicht" beschrieben wird.
Sobald die Kontaktanfragen-Seite aufgerufen wird, wird die Methode searchForRequests() aufgerufen. Die Methode holt mithilfe einer Datenbankabfrage alle Kontaktanfragen, die der Nutzer noch offen hat.

```
1  searchForRequests(id) {
2      this.students = [];
```

```
3       this.accId = id;
4       console.log("accId: " + this.accId);
5       this.ContactRequestTable.filterByValue("receiver", this.accId, "
            contact-request", this.onComplete);
6     }
```

Die Nutzer aus den Kontaktanfragen werden in das Array students gespeichert.
Sobald der Nutzer eine Kontaktanfrage annimmt, wird die Methode accept() ausgeführt. In dieser Methode wird die Kontaktanfrage in der Datenbank aktualisiert und der Nutzer aus dem Array entfernt.

```
1  accept(item: User) {
2       item.json.body.request = true;
3       item.json.body.Zeitstempel = this.ContactRequestTable.TIMESTAMP;
4       this.ContactRequestTable.update(item.json.id, item.json.body, "", (s,
            j) => { });
5       var index = this.students.indexOf(item);
6       if (index > -1) {
7         this.students.splice(index, 1);
8         this.showContactAddedMessage();
9       }
10    }
```

Wenn der Nutzer die Kontaktanfrage ablehnt, wird die Methode removeRequest() ausgeführt, in der die Kontaktanfrage aus der Datenbank und der Nutzer aus dem Array entfernt wird.

3.6 Sprint 6 - Netzwerkübersicht

Das Netzwerk wird in 4 verschiedene Tabs aufgeteilt. Eine für jede Benutzergruppe und einen für alle zusammen. Die Tabs sind sehr ähnlich. Im Konstruktor muss die Liste mit Kontakten gefüllt werden, dazu kann ganz einfach der DataProvider angesprochen werden. Dieser gibt alle Benutzer der entsprechenden Benutzergruppe zurück. Diese werden der property users zugewiesen. Diese wird in der Liste angezeigt. Zuletzt wird noch subscribe() aufgerufen.

```
1  subscribe() {
2          this.notificationService.subscribe(NotificationEvent.
               CONTACT_ACCEPTED, (fromServer, data) => {
3              if (!isPageActive(TabsAll)) {
4                  return;
5              }
6              if (fromServer) {
7                  this.dataProvider.getNewUser(data.sender, data.timestamp)
                       .then(user => {
8                       this.users.push(user);
9                  });
10                 this.notificationService.notify(NotificationEvent.
                       CONTACT_ACCEPTED, false, data);
11             }
12
13         });
14     }
```

Diese Methode ist dafür verantwortlich, dass wenn die Netzwerkseite bereits geöffnet wurde und in dieser Zeit jemand die Freundschaftsanfrage bestätigt, in der Netzwerkanzeige hinzugefügt wird ohne das die Seite neu geladen werden muss. Die Methode fügt beim notificationService das Event hinzu, welches serverseitig ausgelöst wird sobald ein Kontakt dem Netzwerk hinzugefügt wurde. Darauf wird der Nutzer der Liste hinzugefügt
Die Tabs sind vom Grundaufbau alle gleich. Sie unterscheiden sich nur in der Benutzergruppe, welche angezeigt wird.

3.7 Sprint 7 - Profileinstellungen & Accounteinstellungen

WTS-Nutzer haben die Möglichkeit die Sichtbarkeit ihres Profils nach bestimmten Kategorien einzustellen. Diese sind: Persönlich, Email, Adresse und Studium. Die Sichtbarkeit soll nicht nur für bestimmte Nutzergruppen einstellbar sein, sondern auch für Nutzer die nicht im Netzwerk sind. Es gibt jeweils 3 Gruppen innerhalb des Netzwerks und 3 außerhalb. Folgende Abbildung stellt die Datenstruktur in der Datenbank dar:

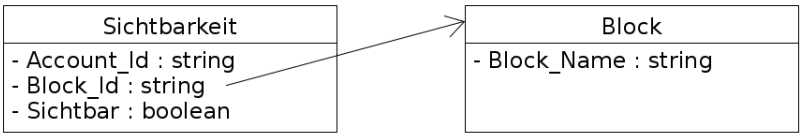

Abbildung 3: Klassendiagramm

Accounteinstellungen

Es gibt 2 Ansichten für die Sichtbarkeitseinstellung: intern und extern. Beide Ansichten sind zwar gleich abgebildet arbeiten jedoch mit unterschiedlichen Daten. Sichtbarkeits Einstellungen werden im folgenden Format in *Block_Name* gespeichert: *intern_address_student* oder *extern_email_uni*. Im Code wird ein JSON-Object erstellt, die diesen Format wiederspiegelt:

```
1  blocks = {
2    personal: {
3      student: undefined,
4      company: undefined,
5      uni: undefined
6    },
7    study: {
8      student: undefined,
9      company: undefined,
10     uni: undefined
11   },
12   address: {
13     student: undefined,
14     company: undefined,
15     uni: undefined
16   },
17   email: {
18     student: undefined,
19     company: undefined,
20     uni: undefined
21   }
22 }
```

Als erstes werden alle Sichtbarkeits-Einträge abgefragt, die die ID des Nutzers beinhalten. Danach werden sie nach Blöcken gefilter, die entweder mit *intern_* oder *extern_* anfangen:

```
load() {
  this.visibilityTable.filterByValue("Account_Id", this.accID, "", (src,
     jsonArray) => {
    jsonArray.forEach(json => {
      if (!json) return;
      this.blockTable.getById(json.body.Block_Id, "", (src, block) => {
      if (!block) return;
      var blockName: string = block.body.Block_Name;
      if (blockName.startsWith(this.prefix)) {
        this.loadValueByString(json, blockName, json.body.Sichtbar);
      }
    });
  });
});
}
```

Die *loadValueByString*-Methode mappt den Namen des Blocks auf das obige JSON-Objekt (blocks). Hier werden alle _ Zeichen ersetzt um den String in das JSON-Objekt umzuwandeln.

```
loadValueByString(json, key: string, value: boolean) {
  var pattern;
  if (this.isExtern) {
    pattern = /extern_/i;
  } else {
    pattern = /intern_/i;
  }
  var obj = key.replace(pattern, '');
  var keys = obj.split('_');
  this.blocks[keys[0]][keys[1]] = value;
  this.data[keys[0]][keys[1]] = json;
}
```

Das JSON-Objekt wird folgendermaßen auf die Oberfläche gebunden:

```
<ion-item>
  <ion-label>{{ 'STUDENTS' | translate }}</ion-label>
  <ion-toggle [(ngModel)]="blocks.personal.student" (ionChange)="
     changeSetting('blocks.personal.student')"></ion-toggle>
</ion-item>
```

Auf diese Weise wird das Ändern der Sichtbarkeitswerte vereinfacht.

Profileinstellungen

Damit die Änderung der Sichtbarkeit in den Accounteinstellungen übernommen wird, muss zunächst in *ProfilePage* die Sichtbarkeit überprüft werden. Als erstes werden Einstellungen für die jeweiligen Kategorien geladen und dementsprechend in Felder initialisiert:

```
this.visibilityService.load(this.accID).then(blocks => {
  blocks.forEach(block => {
    if (block.key == "address") {
    this.addressIsVisible = block.value;
    } else if (block.key == "study") {
    this.studyIsVisible = block.value;
    } else if (block.key == "email") {
    this.mailIsVisible = block.value;
    } else if (block.key == "personal") {
    this.personalIsVisible = block.value;
    }
  });
  this.load();
});
```

Die *visibilityService* holt alle Sichtbarkeitseinträge über die gegebene ID und gibt eine Liste von *key-value-pairs* zurück. Das *key* gibt *Block_Name* zurück und *value* den boolean-Wert.
In der HTML-Datei werden alle felder nach Sichtbarkeitsblöcken sortiert und dementsprechend mit **ngIf* überprüft, ob sie angezeigt werden dürfen oder nicht:

```
<div *ngIf="personalIsVisible">
  <ion-item>
    <ion-label stacked color="wts_blue">{{ 'INTERESTS' | translate }}</
       ion-label>
    <ion-label id="interests"></ion-label>
  </ion-item>
  ...
</div>
```

4 Testen

Im Bereich der Softwareentwicklung dienen Tests dazu, damit überprüft werden kann, ob der geschriebene Code einwandfrei funktioniert und somit die zu Beginn definierten Anfoderungen erfüllt worden sind. In der heutigen Zeit wird meistens „test-driven“ entwickelt. Dies bedeutet, dass zuerst die Tests der einzelnen Methoden, o.Ä., geschrieben werden und erst dann die eigentliche auszuführende Methode.

Diesem Ansatz sind wir nicht gefolgt. Unser Ansatz sah so aus, dass wir (aufgrund von Zeitproblemen) die Anforderungen unserer App genaustens definieren, indem wir dafür einzelne Use-Cases, sowie funktionale und nicht-funktionale Anforderungen für die Applikation aufgestellt haben. Anhand dessen haben wir angefangen zu implementieren. Das Implementierte haben wir stets getestet, indem wir uns jedes Mal aufs Neue die Applikation im Webbrowser oder auf einem iOS bzw. Android Endgerät generieren lassen haben. Mithilfe dessen kann getestet werden, ob die durchgeführte Implementierung funktioniert und somit die notwendige Anforderung erfüllt worden ist. Des Weiteren haben wir innerhalb der Methode auch Konsolenausgaben machen lassen, die uns bei der Implementierung ebenfalls geholfen haben, um festzustellen, ob die Methode einwandfrei funktioniert, oder wo diese festteckt und noch optimiert werden musste.

Mit Abschluss des Projekts lässt sich festhalten, dass wir auch mit der von uns gewählten Methode alle notwendigen Funktionen implementieren konnten und deren Richtigkeit überprüft werden konnte.