Medieninformatik (B. Sc.) - Wintersemester 20/21 - 31.01.2021

Dokumentation zum Semesterprojekt des Moduls Web-Programmierung

Gruppe 07
Benjamin Ansohn McDougall (329956)
Kim Lara Feller (331177)
Louis Grümmer (284033)
Dirk Stricker (333036)

Inhaltsverzeichnis

Einleitung

Aufgabenstellung

Ziel des Moduls Web-Programmierung ist die Erarbeitung einer clientseitigen Webanwendung in Kleingruppen. Das Thema der Anwendung durften wir als Gruppe selbst bestimmen.
Zum Persistieren von Daten ist die im Browser integrierte IndexedDB vorgesehen.

Die Erarbeitung erfolgte in Teilschritten und orientierte sich an folgenden Einsendeaufgaben mit den Schwerpunkten:

1. Entscheidung für ein Thema, Grobkonzeption
2. Prototyp, der Umfang und Funktionalität andeutet
3. Prototyp, in welchem weite Teile der Funktionalität implementiert ist
4. Abgabe des Semesterprojekts

Da die Entwicklung und Versionierung im Team über Git erfolgen sollte, wurde uns ein gruppenspezifisches Repository auf dem GitLab der TH Lübeck zur Verfügung gestellt.
Web-Frameworks und -Libraries durften nicht verwendet werden. Eine Cross-Browser-Optimierung war nicht gefordert.

Technische Daten

Link zum Webspace der Gruppe 07: https://gruppe-07.wp20.mylab.th-luebeck.de/

Weitere Zugangsdaten für unsere Webanwendung werden nicht benötigt.
Es bedarf grundlegend keiner weiteren Erklärung für die Funktionalitäten der Anwendung. Interessant zu erwähnen ist jedoch, dass ein Darkmode für unsere Anwendung über die benutzerdefinierten Farbeinstellungen in den Systemeinstellungen aktiviert werden kann.

Konzeption

Bei dem Thema unserer Anwendung haben wir uns für Projektmanagement entschieden. Die Idee dahinter war es, uns ein Anwendungsziel zu überlegen, dass wir in sinnvolle Einheiten einteilen können. Im gemeinsamen Brainstorming sind uns verschiedene Tools und Funktionalitäten für unsere Anwendung eingefallen, die wir zu einem großen Konzept verknüpfen konnten.

Folgende Projektidee ist während der Bearbeitung der Einsendeaufgabe 1 entstanden:

Unsere Anwendung PROject ist eine Projektmanagement-Anwendung und dient dazu projektspezifische Aufgaben zu verwalten. Mit ihr können Nutzer Projekte anlegen und verschiedene Tools nutzen, um einen Überblick für ihr Projekt zu schaffen. Zum Beispiel ist es möglich, den Projekten Mitglieder zuzuweisen und Teilaufgaben zu verteilen. Der Fortschritt von Teilprozessen bzw. dem gesamten Projekt wird dem Nutzer optisch dargestellt. Eine Kalenderfunktion bietet unter anderem die Möglichkeit Deadlines festzuhalten. Der Nutzer kann somit visuell an Terminabgaben in Form eines Counters erinnert werden. Ein Dashboard bietet eine sinnvolle Übersicht über alle laufenden Projekte. Wahlweise kann sich der Nutzer aber auch nur ein konkretes Projekt oder Inhalte eines angebotenen Tools (wie z.B. offene Aufgaben als to do-Liste) anzeigen lassen.

Fokussierung:

Die Anwendung ist für die Zielgruppe Studierenden gedacht, die ihre (Gruppen-)Projekte für die Universität verwalten wollen.

Angestrebte Features sind:

- Auswahlmöglichkeit der Ansicht
    - Alle Projekte, einzelnes Projekt, einzelnes Tool
- Anlegen von Projekten mit kurzer Projektbeschreibung
- Anlegen von Gruppenmitgliedern
- Anlegen von Aufgaben
    - Zuweisung der Gruppenmitgliedern
    - Visuelle Darstellung von Fortschritten (Aufgaben können abgehakt werden)
- Kalender
    - Termine eintragen
    - Visueller Counter für Deadlines
- Notizfunktion
- Konfigurationsmöglichkeiten Gestaltung
    - Farbgestaltung bei den Projekten
    - Darkmode

Wireframes

Anhand von drei Wireframes haben wir unsere Konzeptidee daraufhin visualisiert.

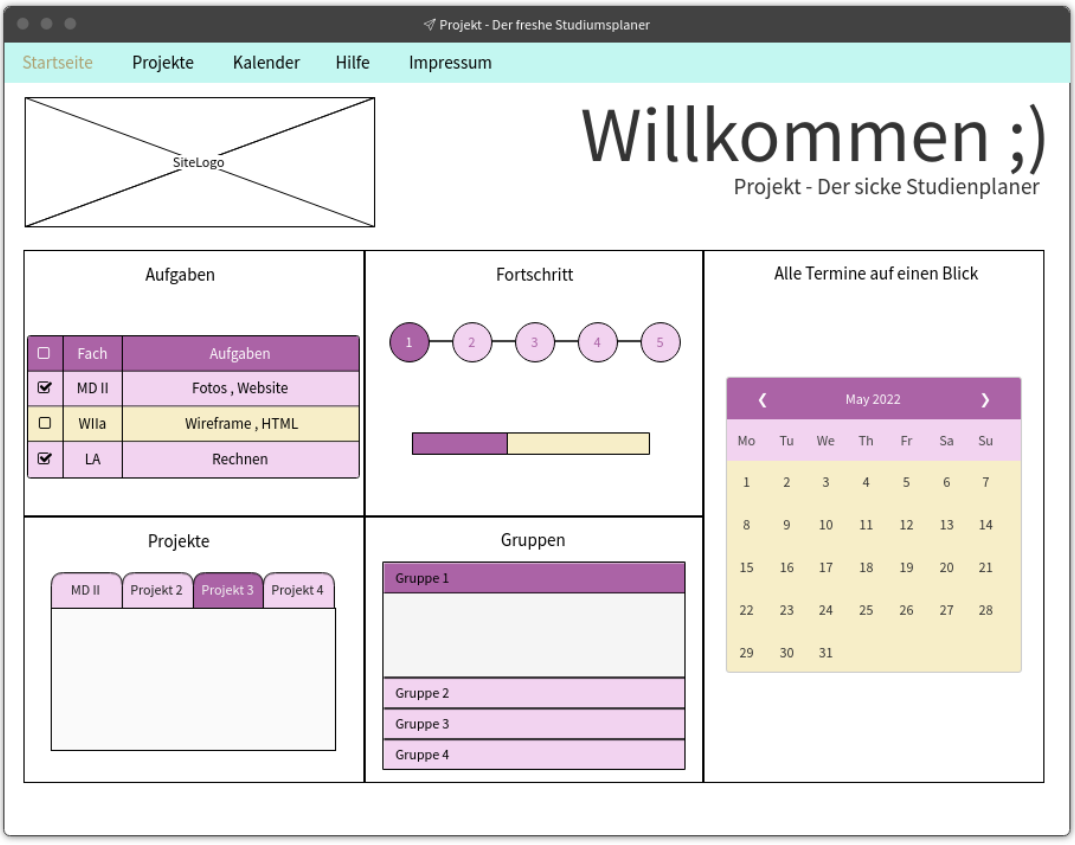


Abb. 1: Wireframe – Startseite

Die Startseite (Abb. 1) zeigt die möglichen Tools des Studienplans. Sie fungiert als Dashboard der Anwendung. Der Nutzende sieht die wichtigsten Informationen zu seinen hinterlegten Projekten auf einen Blick.

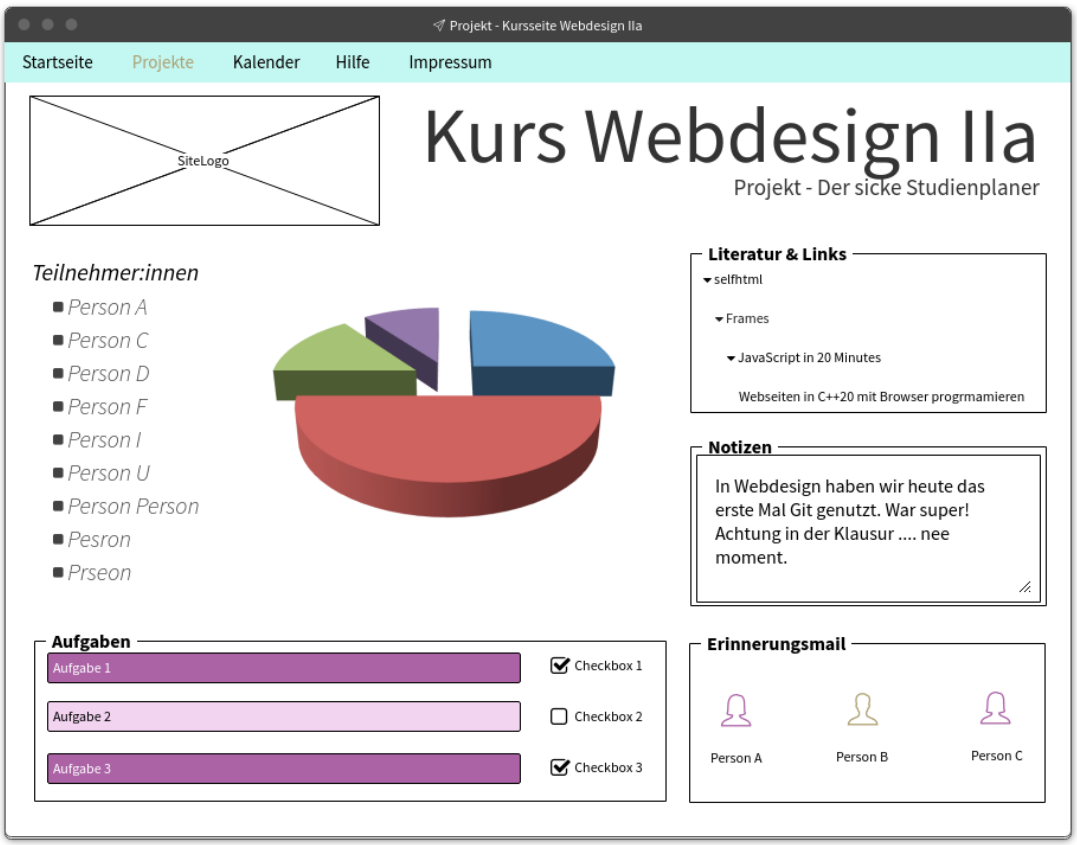


Abb. 2: Wireframe – Projekte

Auf der Projektseite (Abb. 2) erhält der Nutzende dann alle Informationen zu einem konkreten Projekt. Das grundlegende Layout der Projektseite ist für jedes Projekt gleich, um sich möglichst schnell orientieren zu können.
Ergänzend zu dem hier abgebildeten Wireframe haben wir während der Bearbeitung noch drei Button in der unteren rechten Ecke ergänzt. Über diese Schaltflächen kann ein Projekt angelegt, bearbeitet oder gelöscht werden.

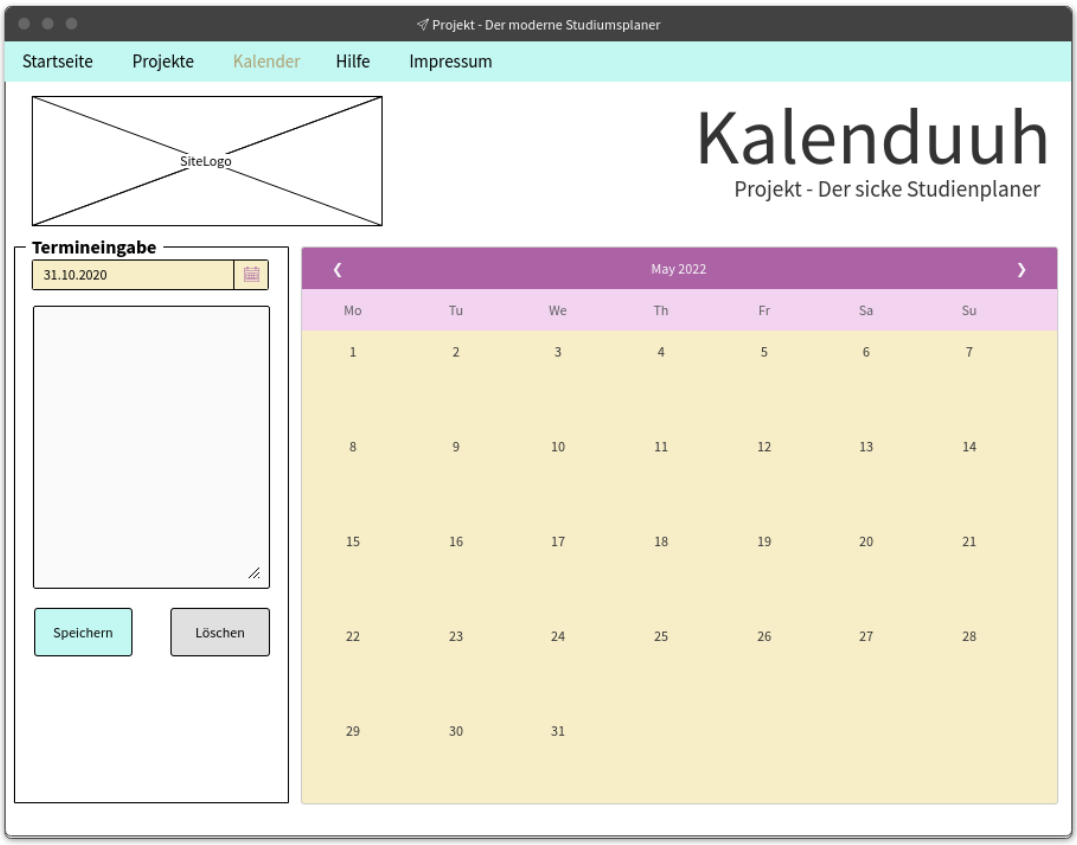


Abb. 3: Wireframe – Kalender

Die Kalenderseite (Abb. 3) zeigt den Kalender, der bereits auf dem Dashboard zu sehen war, noch einmal in einer großen Ansicht. Der Nutzende kann Termine anlegen.

Schwerpunkt der Anwendung

Der Schwerpunkt der Anwendung liegt auf dem Anlegen von Projekten und der Nutzung dessen Daten in verschiedenen Tools, die wir implementiert haben.
Der Nutzende soll mit Hilfe eines Formulars ein Projekt in seinem Studienplaner anlegen können und die Möglichkeit haben, dieses auch im Nachhinein editieren zu können.
Die eingegeben Daten werden daraufhin sinnvoll strukturiert und für verschiedene Funktionen verwendet. Die Mailadressen der ersten drei angelegten TeilnehmerInnen werden z.B. automatisch in die E-Mail-Schnellverknüpfung auf der Projektseite eingebunden.

Datenmodell

Unser Datenmodell basiert auf JavaScript-Objekten, bei denen es sich anbietet, sie als JSON-Objekt in der Datenbank zu speichern. Die Klassen Projekt und Termine sind demnach die wichtigsten Komponenten, die auch in der Persistenzschicht, also der IndexedDb hinterlegt sind. Auch in der Ajax Übertragung der Sprüche, die auf der Kalenderseite angezeigt werden, nutzen wir das JSON Format.

Umsetzung

Code-Organisation

Unser Projekt besteht aus vier html-Dateien: *index* (Startseite), *projekte*, *kalender* und *impressum*. Die dazugehörigen JavaScript- und CSS-Dateien beziehen sich auf verschiedene Teilfunktionen.

Index (Startseite)

Das im Header integrierte Navigationsmenü wurde entsprechend dem Wireframes gestaltet.
Der Reiter Projekte verfügt über eine zweite Hierarchieebene. Das Navigationsmenü wird in der Datei *navbar.js* dynamisiert.
Zur Strukturierung der *index.html* haben wir im *main*-Bereich in einem *article* verschiedene Felder mit Hilfe von *sections* angelegt. Je nach Bedarf befinden sich darin weitere Strukturierungselemente.
In der *section* mit der Überschrift „Aufgaben" werden mit einer Tabelle beispielhaft offene Aufgaben von verschiedenen Projekten angezeigt.
Die *section* „Fortschritt" zeigt beim Anklicken eines Buttons den Fortschritt des jeweiligen Projekts.
An dieser Stelle greifen wir dynamisch auf die in der Datenbank hinterlegten Projekte zu.
Hierbei nutzen wir den Skript Typ „module", um auf die Klasse Projekt zugreifen zu können.
Zudem dynamisieren wir die Fortschrittsanzeige mittels eines Balkens in *progressbar.js*. Diesen Code mussten wir im Verlauf anpassen. Der initiale Skript Typ „*text*" musste durch „*module*" ersetzt werden, damit wir unter anderem die Möglichkeit hatten, die Methode ab Zeile 14 exportieren zu können.

```
14  export function initListeners(projekte) {
15      buttons.forEach( callbackfn: (button : Element , key : number ) => button.addEventListener( type: 'click',  listener: () => {
16          const className = "activeProgress";
17          const isNotButtonAlreadyActive = !button.classList.contains(className);
18          //prüft alle Knöpfe in der buttonClass und setzt den geklickten Button auf active wenn er vorher nicht active war.
19          if (isNotButtonAlreadyActive) {
20              activateButton(projekte[key]._progress, projekte[key]._name);
```

Die Funktion *initListeners* wird in *index.js* aufgerufen und bekommt die angelegten Projekte übergeben. Anschließend haben wir mit einer *forEach*-Schleife einen EventListener (*click*) auf jeden Button gelegt. Den korrekten Umgang dieser Möglichkeit haben wir während der Projektbearbeitung immer weiter verinnerlicht.

Die Konstanten in Zeile 16 und 17 wurden angelegt, um den Code besser warten zu können und um Tippfehler zu vermeiden. Wir prüfen mit einer *if*-Abfrage, ob die im *style.css* implementierte Klasse nicht schon für den jeweiligen Button ein *true* liefert und rufen dann die Funktion *activateButton* auf.

```
35  function activateButton(progress, projektname) {
36      let fillLevel = 0;
37      //setzt ein Intervall und ruft die Funktion fillBar jede Millisekunde auf
38      let run = setInterval(fillBar,  timeout: 1);
39
40      function fillBar() {
41          //erhoeht das fillLevel so lange, bis der übergebende progress um 1 größer ist
42          fillLevel = fillLevel + 1;
43          if (fillLevel === progress + 1) {
44              //beendet das Intervall
45              clearInterval(run);
46          } else {
47              let counter = document.querySelector( selectors: ".counter");
48              counter.textContent = fillLevel + "%" + " " + projektname;
49              fill.style.width = fillLevel + "%";
50          }
51      }
52  }
```

Der Funktion wird der Fortschritt und der Name des jeweiligen Projektes übergeben. Mit Hilfe des *key* kann zwischen den verschiedenen Projekten differenziert werden. Wir haben gelernt, dass key eine ähnliche Funktion wie der Index eines Arrays hat und konnten ihn somit sinnvoll nutzen. Interessant war für uns ebenfalls die in Zeile 38 genutzte Funktion *setIntervall*, die es uns ermöglicht jede Millisekunde die Funktion *fillBar* aufzurufen. Dadurch erzielen wir den visuellen Effekt, dass sich die Fortschrittsleiste dynamisch füllt. Mit einer *if*-Bedingung und der Funktion *clearInterval* stoppen wir das Aufrufen der *fillBar* Funktion.
Bei der *section* „Projekte" mussten wir den Code teilweise überarbeiten, um hier auch dem später verwendeten Skript Typ „module" gerecht zu werden. An dieser Stelle gibt es ein ähnliches Funktionsprinzip, wie bei der *progressbar.js*, weshalb wir uns an dem bereits vorhandenen Code orientieren konnten. Durch das Refactoring haben wir an dieser Stelle die Einbindung der IndexedDB ermöglicht und können die Daten der gespeicherten Projekte nutzen. Gleiches gilt für die *section* „Gruppen".

Projekte

Die Projektseite wurde zunächst streng nach dem in der Planung erstellten Wireframe, sowie ausschließlich statisch entwickelt. Dieses Vorgehen hat den entscheidenden Vorteil, dass mit bereits wenigen Zeilen Code und der ausschließlichen Nutzung von *html* und *css* relativ schnell der Rahmen der Webapplikation implementieren lässt.

Wir nutzten also die Dateien *projekte.html* und *projektStyle.css* für die Implementierung des Sourcecodes. Wir unterteilten auch diesen Teil der Webseite mit einem *header,* der das Logo und Navigationsmenü beinhaltet und dem *main-Abschnitt*, der wiederum in *article* unterteilt ist. In diesen *articles* haben wir dann gegebenenfalls mit *divisions* sowie *fieldsets* noch weitere Unterteilungen vorgenommen.

Mit dem Stylesheet wurde das Design an die Farbauswahl der Webapplikation angepasst und mittels Gridlayout positioniert. Zusätzlich wurden kleinere Spielereien, wie ein drehbares Logo, angelegt.

Die Dynamisierung der Projektseite stand erst im Anschluss im Vordergrund und wurde durch die Einführung der JavaScript-Datei *projektSeite.js* begonnen. Die Entwicklung begann zunächst sehr umständlich und rein funktional. Daher war es nicht verwunderlich, dass die Projektseite zwar einige

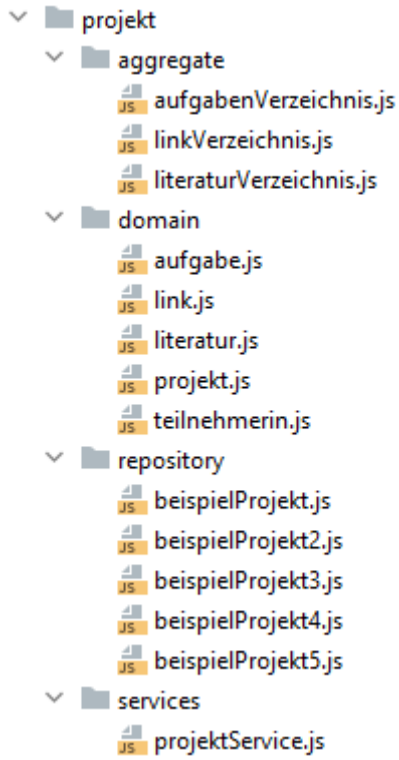
Funktionalitäten vorwies, jedoch die dazugehörige JavaScript-Datei auf über 1500 Zeilen unübersichtlichen Code angewachsen ist, der die Suche nach *bugs* nahezu unmöglich machte. Unnötig bzw. Vergebens war die Entwicklung dennoch keinesfalls. Sei es der Lernprozess oder die Nutzung des entwickelten *JSON-Objekts*, das die Grundlage des objektorientierten Programmieransatzes darstellt.

```
let projektvorlage = {
    //Ausgangswerte für zurückgesetzte/gelöschte Projekte
    //Grunddaten
    "aktiviert"          : false,
    "projektbezeichnung" : null,
    //Teilnehmerdaten <-Sammlung, eventuell in einem bzw. mehreren Arrays zusammenfassen
    "teilnehmer1name"    : "Unbesetzt",
    "teilnehmer1email"   : null,
    "teilnehmer2name"    : "Unbesetzt",
    "teilnehmer2email"   : null,
    "teilnehmer3name"    : "Unbesetzt",
    "teilnehmer3email"   : null,
    "teilnehmer4name"    : "Unbesetzt",
    "teilnehmer4email"   : null,
    "teilnehmer5name"    : "Unbesetzt",
    "teilnehmer5email"   : null,
    "teilnehmer6name"    : "Unbesetzt",
    "teilnehmer6email"   : null,
    "teilnehmer7name"    : "Unbesetzt",
    "teilnehmer7email"   : null,
    "teilnehmer8name"    : "Unbesetzt",
    "teilnehmer8email"   : null,
    "teilnehmer9name"    : "Unbesetzt",
    "teilnehmer9email"   : null,
    //PieChart
    "piechartdone" : 0,
    "piechartdo"   : 0,
    "piecharttodo" : 0,
    //Literatur <-Sammlung, eventuell in einem bzw. mehreren Arrays zusammenfassen
    "literatur1" : "Keine Angabe",
    "literatur2" : "Keine Angabe",
    "literatur3" : "Keine Angabe",
    // Links    <-Sammlung, eventuell in einem bzw. mehreren Arrays zusammenfassen
    "link1href" : null,
    "link1text" : "Keine Angabe",
    "link2href" : null,
    "link2text" : "Keine Angabe",
    "link3href" : null,
    "link3text" : "Keine Angabe",
```

```
40      //Notizen
41      "notizen"   : null,
42      // Aufgaben <-Sammlung, eventuell in einem bzw. mehreren Arrays zusammenfassen
43      "aufgabe1value" : null,
44      "aufgabe1box"   : false,
45      "aufgabe2value" : null,
46      "aufgabe2box"   : false,
47      "aufgabe3value" : null,
48      "aufgabe3box"   : false,
49      "mailperson1"   : "Person A",
50      "mailperson2"   : "Person B",
51      "mailperson3"   : "Person C",
52      personAmail : null,
53      personBmail : null,
54      personCmail : null,
55  }
```

Das auf den beiden Bildern gezeigte Objekt wurde durch die Schaffung einer Ordnerstruktur sowie die Umsetzung des Konzepts der objektorientierten Programmierung in übersichtliche und leicht zu wartende Teile abstrahiert.

So ergibt sich die folgende Ordnerstruktur:



Im Ordner *aggregate* fanden die drei Klassen für die Sammlungen der Aufgaben, Links sowie Literatur ihren Platz und nehmen jeweils die passenden Objekte aus dem *domain* Ordner in ein Array auf.

Vereinigt bzw. instanziiert werden diese Klassen in der *projekt.js*:

```
export class Projekt {
    _nummer;
    _aktiviert;
    _name;
    _teilnehmerListe;
    _literaturVerzeichnis;
    _notizen;
    _links;
    _aufgaben;
    _pieSize;
    _progress;

    constructor(nummer : number  = Math.round( x: Math.random() * 100000000), aktiviert : boolean  = true, name : string  = 'Klick hier',
                teilnehmerListe : Teilnehmerin[]  = [new Teilnehmerin()],
                literaturverzeichnis : Literatur[]  = [new Literatur()],
                links : Link[]  = [new Link()],
                notizen : string  = 'Notizen zum Projekt',
                aufgaben : Aufgabe[]  = [new Aufgabe()],
                pieSize : number[]  = [500,400,300],
                progress : number  = 1) {
        this._nummer = nummer;
        this._aktiviert = aktiviert;
        this._name = name;
        this._teilnehmerListe = teilnehmerListe;
        this._literaturVerzeichnis = literaturverzeichnis;
        this._links = links;
        this._notizen = notizen;
        this._aufgaben = aufgaben;
        this._pieSize = pieSize;
        this._progress = progress;
    }
}
```

Hier zeigt sich deutlich der konzeptionelle Ursprung der Klasse(n). Jedoch war die Möglichkeit Objekte vom Typ Projekt zu erzeugen noch nicht die endgültige Lösung, um diese auch auf der Webseite anzeigen zu lassen. Daher musste noch eine dynamische Verbindung von der Klasse Projekt und der Webseite geschaffen werden, die wiederum alle anderen dazugehörigen Klasse instanziiert.

Diese Verbindung wird durch die Klasse *projektService.js* im Ordner */services* realisiert. Hier wird zum einen durch die Methode *fillWindow()* die Schnittstelle zwischen den Projekt-Objekten und der *projekt.html* vereinbart, zum anderen durch die Methode *fillForm(),* in der eine Verbindung der Projekt-Objekte mit den Elementen des Formulars implementiert ist.

Seitens der Zuordnung zu den HTML-Elementen haben wir *DOM - Document Object Model* gewählt und manipulieren so die Werte des passenden HTML-Elements.

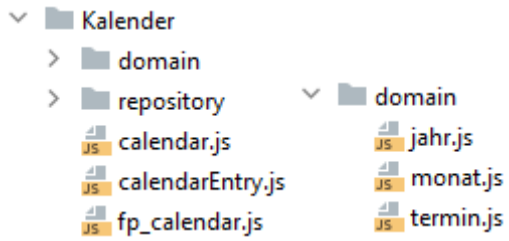
Zusätzlich mussten aber auch innerhalb der Methode *fillWindow()* darauf geachtet werden, dass die *listitems* der HTML-Seite wieder auf den Anfangswert gesetzt werden, damit es zu keinen fehlerhaften Einträgen beim Durchwechseln der Projekte durch Überlagerung kommt.

Nach dem diese Methoden, sowie Klassen des Projekts entwickelt wurden, konnte die Überarbeitung von *projektSeite.js* beginnen. Dafür wurden neben weiteren Methoden unter anderem die E-Mail-Versendung an die neuen Gegebenheiten angepasst oder neue Methoden geschaffen.
Ein Durchschalten durch die Projekte mittels Mausklick oder die Nutzung der Pfeiltasten auf der Tastatur waren nun möglich. Der *keydown*-EventListener, der bereits durch das Impressum vorhanden war, ist nur eines der Beispiele, wie wir versucht haben, bereits bestehenden Code zur Lösung einzubinden.
Neue Projekte können mit einem Formular auf der Seite *projekte.html* erstellt werden. Auch ist es an dieser Stelle möglich, bereits bestehende Projekte nachträglich zu bearbeiten, wenn beispielsweise weitere TeilnehmerInnen hinzugefügt werden sollen oder sich hilfreiche Links mitsamt der Beschreibung ändern.

Kalender

Die Gestaltung des Header-Bereichs ist auf der identisch zu der Startseite. Das gilt für die HTML- und CSS-Elemente. Der Aufbau des Kalenders ist ebenfalls von der Startseite übernommen und nur geringfügig in der Größe angepasst. Somit konnten wir schon vorhandenen Code für beide Seiten nutzen. Zu der dynamischen Darstellung und Generierung der Monatsnamen und den entsprechenden Tagen sind noch weitere Funktionen dazu gekommen.
Neu hinzugekommen ist unter anderem die linke Seite im Bereich *main*. Diesen Bereich haben wir für die Einträge im Kalender, also die Erfassung der Termine vorgesehen.
Die Erstellung eines Kalenders haben wir initial genutzt, um die neueren OOP-Elemente in EcmaScript 6 auszuprobieren. Zudem wurde die Struktur des Codes in diesem Bereich stark an Domain-Driven-Design angelehnt. Letzteres kam zum Einsatz, um den Aus- und Umbau des Codes so flexibel und nah an der fachlichen Domäne zu halten wie möglich. Somit können wir uns auch zusätzliche Code-Dokumentation sparen, da die Klassen so nah an der Fachlichkeit des Kalenders modelliert sind, dass die Verwendung der einzelnen Felder und Klassen selbsterklärend ist.
Weiterhin bedienen wir uns in diesem Bereich am Skript Typen „module“. Somit haben wir die Möglichkeit, die verschiedenen Klassen zu exportieren und somit in anderen Modulen darauf aufbauend zu entwickeln. Als Beispiel ist hier die Klasse *Termin* zu nennen, die von der Klasse *Monat* und *calendarEntry.js* verwendet wird.

Folgend die Projektstruktur des Kalenders auf Ordnerebene:



Die Architektur ist an dieser Stelle noch überschaubar, da nicht viele Klassen nötig sind, um die Fachlichkeit des Kalenders adäquat festzuhalten.

Die Domäne beinhaltet die Klassen Jahr, Monat und Termin. Sie beschreiben die fachlichen Werte, die der Domäne des Kalenders zugrunde liegen. *Termin.js* wird derzeit genutzt, um einen möglichen Termin festzuhalten. Durch die Erstellung der Klasse *Termin* ist es uns möglich, diesen Termin so nah wie möglich an der Fachlichkeit zu modellieren. Somit fiel es uns leicht, ein Formular zu erstellen und festzulegen, wie wir uns einen Termin genau vorstellen. Weitere Dokumentation oder Erläuterung im Code ist dadurch nicht notwendig.
Bei den Klassen *jahr* und *monat* sind wir äquivalent vorgegangen. Sie sind für erweiterte Darstellungsmöglichkeiten vorgesehen, kommen aktuell aber nicht zum Einsatz. Der ursprüngliche Plan bestand aus verschiedenen Operationen für gesamte Monate bzw. Jahre anzubieten.

Wir haben ein kleines Gimmick durch die Verwendung von Ajax eingebaut. Verschiedene Sprüche werden aus einer auf dem Server hinterlegten Textdatei per Zufall geladen und dann im Notizfeld des Termins als Platzhalter hinterlegt.

Die Klasse *calendarEntry.js* kümmert sich um das Eintragen eines Termins in den Kalender und die Verarbeitung (CRUD - Operationen), sowie die Erfassung eines neuen Termins über die *kalender.html*.

Somit wäre eine weitere Teilung der Klasse *calenderEntry* möglich. In Bezug auf ihre Verantwortlichkeiten ist sie deutlich überladen, allerdings noch überschaubar. Daher haben wir uns vorerst gegen ein Refactoring entschieden.

Calendar.js war die ursprüngliche Version des eigentlichen Materials als Objekt-orientierte Lösung. Jedoch interessierte uns der „funktionale“ Aspekt von JavaScript und die neueren Hilfsfunktionen für Arrays und Objekte. Diese wollten wir ausgiebig testen. Das ist uns in der *fp_calendar.js* Datei auch gelungen, jedoch litt darunter die Lesbarkeit und somit die Verständlichkeit des Codes. Das haben wir durch die Nutzung von möglichst sprechenden Variablen- und Funktionsnamen versucht zu kompensieren, werden aber im Verlauf dieses Abschnitts noch einmal auf die Problematik eingehen. Vorteilhaft an der funktionalen Implementierung ist, dass mit weniger Zeilen Code mehr Funktionalität realisiert werden kann. Ein funktionaler Ansatz ist also effizienter.
Zudem haben wir versucht Seiteneffekte zu vermeiden und fast ausschließlich mit Konstanten zu arbeiten. Daher auch die Verwendung von Funktionen mit so wenig Seiteneffekte wie möglich, was sich bei der Gestaltung eines graphischen Frontend nicht vollständig vermeiden lässt.

Impressum

Wir haben in der Konzeptionsphase im Navigationsmenü ein Impressum eingeplant. Würde man unsere Anwendung tatsächlich veröffentlichen, wäre ein Impressum eine gesetzlich vorgeschriebene Angabe, da es einen Verantwortlichen für die Inhalte der Seite geben muss.

Grundlegend hatten wir nicht geplant, diese Seite großartig zu gestalten. Da während der Bearbeitung des Semesterprojektes jedoch viel recherchiert wurde, sind wir alle immer wieder auf interessante Beiträge und Funktionen bzw. Methoden gestoßen, die wir gerne testen wollten. Das ausprobierte Tutorial für unseren Hintergrund (die bunten Kugeln) ist eines dieser Beispiele. Gemeinsam haben wir uns dafür entschieden solche Spielereien, die den Lernprozess fördern, mit einzubinden und unsere Anwendung damit aufzuwerten. Da die Impressum Seite ansonsten sehr leer erschien, wurde der Versuch eines Mini-Spiels auf dieser Seite eingebunden.

Die *impressum.html* besteht zunächst aus dem bereits bekannten Kopfbereich mit dem Navigationsmenü, Logo und dem Seitentitel.
Im *body* ist dann als erstes das *canvas*-Element und die Einbindung des *canvas.js* zu finden, damit auch das Impressum den bereits erwähnten Hintergrund erhält.
Die Seite ist in drei *article* unterteilt: der erste *article* umfasst das eigentliche Spielfeld, wofür ebenfalls ein *canvas*-Element zum Einsatz kommt. Der zweite *article* zeigt ergänzende Spielinformationen und der dritte *article* zeigt einen Mustervorlage, die für eine Impressumsangabe genutzt werden kann.
Ergänzend zum *header.css* ist die Gestaltung der Seite mit CSS im *impressum.css* zu finden.

Das Spielfunktion wurde im impressum.js implementiert. Für das Spielfeld wird ein canvas genutzt. Dies ist ein Bereich, in den anhand von Höhen- und Breitenangaben per JavaScript gezeichnet werden kann. Es ist also vergleichbar mit einem Koordinatensystem.
Sobald das HTML-Dokument vollständig geladen ist (EventListener auf DOMContentLoaded), wird ein Takt erzeugt, um die Funktion *taktung* anhand dessen wiederholt aufzurufen. Desweiteren wird das *canvas* als Spielfeld definiert und ein neues Objekt für die Spielfigur *papierflieger* erzeugt, dem ein Bild zugewiesen wird, was letztlich als Figur dient. Über die *drawImage*-Methode wird dieses Bild dann entsprechend der Koordinaten (Variablen sinngemäß mit x und y bezeichnet) ins *canvas* „gezeichnet“.

Die Funktion *zeichneZiel* erzeugt äquivalent hierzu die zu erreichende Zielfigur. Die Position des Ziels wird über die JavaScript Funktion *Math.random* zufällig erzeugt. Da eine Gleitkommazahl zwischen 0-1 erzeugt wird, wird der Wert über *Math.floor* gerundet. In der Funktion *zielErreicht* kann somit überprüft werden, ob sich die Spielfigur an der gleichen Position wie die Zielfigur befindet und der Spielstand hochgezählt werden soll.
Die Spielfigur wird mit Hilfe der Pfeiltasten bewegt. Hierzu wird ein EventListener genutzt, der auf *keydown* reagiert. Der jeweilige *key* manipuliert die entsprechenden Variablen der Figur. Damit diese Veränderung zu sehen ist, muss die Spielfigur regelmäßig an der neuen Position wieder eingezeichnet werden. Hierzu dient die Funktion *taktung,* die genau das im bereits erwähnten Takt macht. Sobald die Spielzeit abgelaufen ist, wird die Funktion *beendeSpiel* aufgerufen, die den Taktgeber zurücksetzt.

Aktuell kann das Spiel nur durch ein Neuladen der Seite erneut gestartet werden.

Anforderungen

Anforderungen zum Umfang

Mindestens 3 Ansichten mit verschiedenen Strukturelementen/Features
Es gibt vier Seiten, die auf folgenden html-Dateien basieren:
- index.html
- projekte.html
- kalender.html
- impressum.html

Mindestens 1 Formular
Formular zum Anlegen und Bearbeiten von Projekten in *projekte.html* (Zeile 173 bis 283) mit *projektSeite.js.*

Anforderungen zum Layout und Design

Seiten enthalten Kopf-, Haupt- und Fußbereich
Alle Seiten sind mit den HTML-Tags *header*, *main* und *footer* strukturiert

Navigationsmenü
Auf allen Seiten ist im *header* ein Navigationsmenü implementiert, gekennzeichnet durch den HTML-Tag *nav.*
Beispielhaft zu finden in *index.html* ab Zeile 26 bis Zeile 63

Mindestens eine Tabelle
Eine Tabelle für die Aufgaben befindet sich auf der Startseite (*index.html Zeile* 81, *style.css* Zeile 59). Ebenfalls basiert der Kalender auf einer Tabelle (z.B. *index.html* Zeile 177, *kalenderStyle.css*, *jahr.js, monat.js, termin.js, fp_calendar.js*).

Zwei- oder drei-spaltige Bereiche (nicht durch Tabelle realisiert)

Verwendung von CSS Grid (z.B. *index.html*, *css.style* Zeile 44)

Grundlegendes Layout und statische Effekte
Layout wurde getreu der Wireframes umgesetzt, auf der *projekte.html* durch weitere Buttons ergänzt.

Dynamische Effekte mit CSS
Die :hover CSS Pseudoklasse wird an mehreren Stellen genutzt, um dynamische Effekte zu erzielen.

Zum Beispiel verändert sich die Farbe der Schrift, sobald über eine Referenz im Navigationsmenü hovert (*style.css* Zeile 189)

Auch unser Logo wird durch die Nutzung der :hover Pseudoklasse um 360° um die eigene Achse gedreht und somit dynamisiert (*projektStyle.css* Zeile *20).* Des Weiteren verändert sich beim Überfahren des Projektnamens und des E-Mail-Kontaktbildes der Mauszeiger (*projektStyle.css* Zeile 20*)*.

Zwei Hierachieebenen im Navigationsmenü
Auf allen Seiten implementiert, exemplarisch in *index.html* ab Zeile 26
Eine zweite Hierarchieebene ist beim Reiter "Projekte" implementiert.

Ausgearbeitetes Layout passend zum Konzept / Nutzung von Flexbox bzw. anderweitige Designbesonderheiten
Implementierung orientierte sich sehr stark am Wireframe, daher wurden viele verschiedene Elemente zur Gestaltung des Layouts verwendet.
Um z.B. das zugrundeliegende Layout der Startseite und Projektseite umzusetzen, wurde ein CSS Grid verwendet (in *style.css* Zeile 44 sowie *projektStyle.css* zu sehen)
Die Tabs auf der *index.html* für die Projekte wurden dahingehend mit einer Flexbox positioniert (*style.css* 171).

<u>Anforderungen an die Funktionalität</u>

Formular zur Speicherung und Veränderung von Daten, min. HTML5-Validierungsmöglichkeiten; Formular enthält zusätzliche Elemente, Validierungen mit JavaScript;
Echtzeitvalidierung von Formularen, Nutzung von Formulardaten für Anwendungslogik
Formular zum Anlegen und Bearbeiten von Projekten auf der Projekteseite (*projekte.html* ab Zeile 191, *projektSeite.js* ab Zeile 342)

JavaScript zur Dynamisierung von Interaktionen, Umsetzung elementarer Logik, Verarbeitung von Formular-Eingaben
Die Inhalte auf der *projekt.html* werden anhand der Methode *fillWindow()* in der Klasse *projectService.js* dynamisch erstellt (Zeile 16).

Die Inhalte im Formular auf der *projekt.html* werden durch die Nutzung der Methode *fillForm()* in der Klasse *projectService.js* dynamisch gefüllt (Zeile 83).

JavaScript-Einsatz für Anwendungslogik
Javascript wurde genutzt, um Klassennamen zu manipulieren, die z.B. das Anzeigen und Ausblenden des Formulars ermöglichen, Einsetzen von verschiedenen EventListener

Komplexere JavaScript-Anwendungslogik
Spiel auf der Impressum Seite (*impressum.js*), *fp_calendar.js,* Verwendung von *canvas*-Elementen, IndexedDb Operationen

Verwendung von Ajax zum Auslesen und verarbeiten von JSON-Objekten
Siehe Methode *SpruecheDesTages* in *calendarEntry.js*

JSON-Daten formatiert ausgeben
Wird ausgiebig auf der *projekte.html* genutzt.
projektService.js füllt die Projektdaten aus der Datenbank in die Projektseite.

JSON-Daten für Anwendungslogik nutzen
Wir nutzen JSON-Daten aus der Datenbank, um die Progressbar zu füllen (*progressbar.js*).

IndexedDB: Daten anlegen und auslesen
Die Projekte, auf denen unsere Anwendung bzgl. der Daten basiert, werden in der IndexedDB "WPGruppe07" gespeichert (*indexedDB.js*, *index.js Zeile 99*, *projektSeite.js* Zeile 231) und wiederum aus der Datenbank für verschiedene Tools zur Verfügung gestellt.

IndexedDB: mehrere Collections, Suchen, Löschung von Einträgen
Termine und Projekte sind mit einem Index versehen und sind zwei verschiedene ObjectStores.
Wir nutzen die Funktionen *get()*, *CursorWithValue (hier liegt ein Beispiel vor)* und *getAll()*, um Daten zu suchen oder eine Teilmenge von Daten zu laden. Auf der *projekte.html* kann man die Projekte von der IndexedDb löschen.

Beispiele

Implementierung des Darkmode

Für die Implementierung des Darkmode haben wir die *@media (prefers-color-scheme: dark)* Regel im CSS verwendet und in dieser unsere Farben für den Darkmode definiert. So können wir die vom Nutzer in den systemweiten Einstellungen ausgewählte Personalisierung nutzen. Der Anwender befindet sich beim Öffnen der Webseite dadurch direkt in seinem präferierten Darstellungsmodus und ist nicht gezwungen, dies separat über einen Switch zu togglen.

Beispiel einer IndexedDb Operation

An dieser Stelle würden wir gerne auf die Implementierung der IndexedDb eingehen. Die Kernlogik liegt in der Klasse IndexedDB, die sich in der gleichnamigen Datei befindet im Ordner *~/public/js*.
Da wir an dieser Stelle nicht auf alle Details eingehen wollen, konzentrieren wir uns auf einen Teil der Methode *initialize*:

```
// Abfangen ob die Verbindung erfolgreich war.
requestToOpenDb.onsuccess = () => {
    const db = requestToOpenDb.result;
    console.log(`IndexedDb ${this.dbName} mit der Version ${this.dbVersion}: ${db.name}`);
    resolve(db);
};
```

Auf jeder Seite, die Informationen von der IndexedDb lesen möchte, z.B. die Start- oder Projektseite, wird als erstes die *initialize* Methode am zuvor erstellten IndexedDb Objekt aufgerufen. Damit wollten wir auf den einzelnen Seiten Code-Duplikate verhindern und die Operationen, die an der IndexedDb ausgeführt werden, in der Klasse bewältigen. Somit können wir die einzelnen Seiten weitestgehend von technischen Details freihalten.

Das Ergebnis des Callbacks haben wir dann direkt in den nach den Seiten benannten Dateien verarbeitet. Hier ein Auszug aus der *index.js*:

```
99  const nProjekteAusDbLaden = (anzahlZuHolendenProjekte) => {
100     const indexedDb = new IndexedDB();
101     indexedDb.initialize().then((db) => {
102         const projektVerzeichnis = [];
103         const cursorRequest = indexedDb.retrieveItemsWithCursor(db);
104         cursorRequest.onsuccess = (event) => {
105             const cursor = event.target.result;
106             if (cursor && projektVerzeichnis.length < anzahlZuHolendenProjekte) {
107                 projektVerzeichnis.push(cursor.value);
108                 cursor.continue();
109             } else {
110                 initListeners(projektVerzeichnis);
111                 gruppenAusfuellen(projektVerzeichnis);
112                 tabsAusDbFuellen(projektVerzeichnis);
113             }
114         }
```

In der zweiten Zeile des Codes (Zeile 100) wird die IndexedDb initialisiert und in der darauffolgenden auf einen Callback aus der *initialize* Methode gelauscht (Zeile 101). Somit fangen wir den IDbRequest, aus einen Promise Objekt, direkt ab, wenn diese erfolgreich angefragt wurde. In dieser Methode wollen wir nur eine bestimmte Anzahl von Projekten aus der IndexedDb laden. Potentiell könnte eine große Anzahl an Projekten abgelegt sein. Um Zeit zu sparen, nutzen wir einen Cursor (Zeile 103 und 105) um die Anzahl an Schritten durch das ObjectStore zu iterieren, die als Parameter in der Signatur der Methode angegeben ist. Die so angefragten Projekte werden zur schnelleren Nutzung im lokalen Speicher erfasst: im Array *projektVerzeichnis* (Zeile 107). Nachdem die entsprechenden Projekte ermittelt worden sind, werden die Elemente auf der *index.html* mit den Daten aus der IndexedDb initialisiert und an den entsprechenden Stellen im HTML, bzw. CSS Code eingefügt (Zeilen 110 - 112). Wir möchten darauf hinweisen, dass wir hier nur einen kleinen Ausschnitt des Codes ausgewählt haben und entsprechende Methoden zur Fehlererfassung (z.B. *onerror*) vorliegen, falls die Anfragen fehlschlagen, was nicht aus unserer Auswahl ersichtlich wird.

Schwierigkeiten

Schwierigkeiten traten, wie bereits bei dem Punkt Umsetzung erwähnt, bei der Dynamisierung der Projektseite auf. Viele Versuche einer Dynamisierung mit relativ begrenzten Wissen der meisten Gruppenmitglieder, scheiterten Aufgrund des Code Umfangs und umständlicher Methoden. Zusätzlich wurden viele Stunden des Codings nicht mit einem Commit abgeschlossen, sondern verworfen.

Reflexion

Unser initiales Wireframe hat uns retrospektiv sehr eingeschränkt und war an vielen Stellen auch komplett unlogisch. Für die Zukunft würden wir auf der *index.html* nicht mehr 5 Fortschrittselemente, 3 Aufgaben, 4 Projekt Tabs und 4 Gruppen anzeigen lassen.
Weiterhin ist das Hin- und Herschalten zwischen den Projekten nicht intuitiv und auch nicht im Wireframe berücksichtigt. Zwar können wir durch die linke und rechte Pfeiltaste und das Klicken auf die Projekttitel zwischen den Projekten schalten, in Bezug auf Usability Engineering empfinden wir das allerdings als suboptimal.

Auf der Kalenderseite sind viele offene Tasks, die kurz vor der Vollendung stünden. Unter anderem steht der Umbau der Persistenz vom LocalStorage zur IndexedDb bevor und die Verknüpfung von Projektaufgaben und deren Deadlines an die entsprechenden Termine. Wir hatten den Hauptfokus unserer Anwendung auf die gute Nutzbarkeit der Projekt- und Startseite gelegt. Insbesondere die Anbindung an die IndexedDB und die Verdrahtung der Daten an die View, also die *projekt.html* und *index.html*. Zudem läge vor dem Ausbau der Kalenderseite ein großes Refactoring des Codes der Projektseite an, da die *projekt.js* sehr groß und teilweise unübersichtlich geworden ist. Dies ist auch der Grund, weshalb uns die zeitlichen Ressourcen zur Vervollständigung des Kalenders gefehlt haben.

Vorwissen

Folgend ist das persönliche Vorwissen jedes Gruppenmitgliedes beschrieben:

Benjamin Ansohn McDougall
Seit den frühen Tagen des öffentlich verfügbaren Internets schon an der Erstellung von Internetseiten interessiert. Zuerst eher statische Seiten (ab HMTL 2) mit Frames bis hin zu Adobe Flash Seiten mit einigen animierten Szenen. Dynamische und serverseitige Verarbeitung von Daten anhand von JavaScript kam jedoch deutlich später. Hauptberuflich bin ich seit 13 Jahren Softwareentwickler, davon jedoch ca. 5 Jahre mit einem Schwerpunkt auf Web-Programmierung, der derzeit durch Frameworks wie Angular wieder stark in den Fokus rückt.

Kim Lara Feller
Durch zuvor belegte Module waren mir einige Begriffe geläufig, jedoch konnte ich im Bereich Web-Programmierung keine praktischen Erfahrungen vorweisen. Durch erste Programmiererfahrungen in Java hatte ich jedoch eine Vorstellung davon, was mit JavaScript möglich ist.

Louis Grümmer
Mir waren zwar einige Begriffe wie HTML, CSS und JavaScript bekannt und durch zuvor belegte Module wie Grundlagen der Programmierung 1, hatte ich eine Vorstellung wie JavaScript funktionieren könnte. Ich hatte allerding vor dem Modul noch keinen Kontakt oder Erfahrung zur Web-Programmierung.

Dirk Stricker
Meine erste Internetseite habe ich mittels Netscape Composer durch Nutzung der Drag and Drop Funktionen erstellt. Mir waren vor Modul Beginn einige, aber doch eher veraltete, Begrifflichkeiten bekannt. Sonst hatte ich keinerlei Vorerfahrungen mit der Web-Programmierung.

Organisation

Die Organisation innerhalb der Gruppe wurde folgendermaßen organisiert:

Für die Entwicklung der Projektidee, Gestaltung der Wireframes sowie Dokumentation von Ergebnissen und Austausch zu größeren Meilensteinen trafen wir uns regelmäßig via Videochat auf Skype. Dort besprachen wir auch die auftretenden Probleme und versuchten gemeinsam Lösungen zu finden, wenn einer von uns nicht weiterkam. Darüber hinaus planten wir die kommenden Paare für das Paarprogrammieren und deren sogenannten Sprintziele. Uns war es ein Anliegen, dass keiner das Gefühl hat, allein arbeiten zu müssen. So konnten wir uns gegenseitig optimal unterstützen und beispielsweise Tippfehler vermeiden.
Für das Paarprogrammieren nutzten die Partner überwiegend Skype mit der Voice Chat-Funktion und der Möglichkeit den Bildschirm in HD zu übertragen. Dabei programmierte einer in der IDE und der andere formulierte den Code und recherchierte Lösungsmöglichkeiten und kontrollierte die Eingaben

des anderen.
Bei Problemen, die hin und wieder auftraten und nicht innerhalb eines Paares gelöst werden konnten, sowie zum Austausch über zwischenzeitliche Arbeitsstände, erstellten wir eine Chatgruppe mit der App “Signal”.

Lernerfolg

Die wichtigsten drei Aspekte, an denen wir unseren Lernerfolg erkennen, sind folgende:

- Erarbeitung des Lehrmaterials und dessen Anwendung im Projekt
- Grundlegende Einblicke in die Frontendentwicklung
- Gemeinsam als Team an einem Entwicklungsprojekt arbeiten

Klassen in JavaScript einsetzen zu können war für einige Teammitglieder neu, da sie entweder nur *function* oder *prototype* kannten. Es hat letztlich sehr viel Spaß gemacht, Klassen einsetzen zu können, um ein wenig Struktur in unsere Anwendung zu bekommen. Dadurch fiel uns die gesamte Dynamisierung unseres Projekts leichter und wir konnten Erfolge erzielen.

Ein wichtiger großer Lernerfolg war die Festigung im Umgang mit *higher-order-functions*. Da dies kein Hauptfokus dieser Veranstaltung ist und auch nicht für die Bewertung relevant, haben wir es nicht durchgängig umsetzen können. Unser Verständnis vom Einsatz der Methoden *reduce*, *map* und anderen neueren Array Methoden, sowie Kurzschreibweisen - z.B. [...object, einFeld] - hat sich sehr gebessert und die Denkweise beim Programmieren erweitert. In Zukunft werden wir weiter mit diesen Operationen arbeiten.
Zudem hat der Umgang mit der IndexedDb unser Verständnis von asynchronen Operationen in JavaScript stärken können. Interessant dabei war, dass wir ursprünglich eine falsche Herangehensweise hatten und diese mit der Übung korrigieren konnten. Nach vielen Fehlversuchen haben wir einen Weg gefunden auch hier unsere Denkweise anzupassen und zu verstehen, wo und vor allem wann, welche Funktionsaufrufe passieren. Promises und Events haben wir ausschließlich verwendet, um asynchronen Code zu bezähmen.

Negativ war, dass wir es nicht wirklich geschafft haben uns außerhalb von Messenger-Diensten und Meetings zu organisieren. Zwar war der initiale Versuch der Nutzung von User-Stories da, jedoch zeigte sich sehr schnell, dass die Mehrheit des Teams mit einer solchen Vorgehensweise (noch) nichts anfangen konnte. Somit reagierten wir eher, als das wir agierten und häufig kam die Frage auf, was denn noch zu tun sei und vor allem wo. Dies hätte man wahrscheinlich mit fest verteilten Aufgaben zum Großteil vermeiden können. In den Meetings ist daher vor allem am Anfang sehr viel Zeit verloren gegangen, da eine klare Richtung fehlte. Meistens fanden wir diese nach den Meetings immer wieder aufs Neue, möchten in Zukunft aber für vergleichbare Projekte definitiv daran arbeiten.