T2 Gruppenprojekt

ON-Manager

07.05.2021

Jonas Bloching, Tobias Huber, Linda Schäfer,
Laura Schramm, Dominik Thurau und Tabea Wöhr

https://github.com/DominikThurau/Stencil-Group-Project-21

Inhaltsverzeichnis

1. Komponenten **3**
1.1 Kalender (Jonas) **4**
Idee 4
Reflexion 5
1.2 To-Do-Liste mit verschiedenen “Blättern” (Laura) **6**
Idee 6
Reflexion 7
1.3 Zeichenfläche (Dominik) **8**
Idee 8
Reflexion 9
1.4 Zufallsgenerator Gruppen (Linda) **10**
Idee 10
Reflexion 11
1.5 Start-Screen (Tabea) **12**
Idee 12
Reflexion 13
1.6 Footer (Tobias) **15**
Idee 15
Reflexion 16

2. Reflexion Gruppenarbeit **17**

1. Komponenten

Der ON-Manager ist eine App für Studierende des Studiengangs Onlinemedien an der DHBW in Mosbach. Sie soll die Studierenden bei ihrem studentischen Alltag unterstützen und ihnen hilfreiche Funktionen bieten.

Unser Design orientiert sich an dem Ansatz “mobile first”, der volle Funktionsumfang ist bisher jedoch lediglich in der Desktop-Variante der Anwendung verfügbar.

1.1 Kalender (Jonas)

Idee

Besonders in den Theoriephasen ist es sehr wichtig, alle seine Termine im Überblick zu behalten. Egal ob Vorlesungen, Gruppenmeetings, Abgaben und andere Prüfungsleistungen, einen Terminkalender sollten Studierende sorgfältig führen (auch wenn er nur dafür sorgt, dass man weiß ab wann die Vorlesung am nächsten Tag endlich vorbei ist und man sich mit seinen FreundInnen treffen kann).

Der Kalender soll übersichtlich den gesamten Monat darstellen und es sollen Termine eingefügt und entfernt werden können. Bei der Entwicklung des Prototyps hatte ich nicht bedacht, dass wir eine OnePage-Webapplikation entwickeln und keine direkte App für ein Smartphone. Im Prototype *(siehe Abbildungen)* war vorgesehen die Monate (scrollbar) untereinander anzuzeigen. Zum Hinzufügen von Terminen war ein Plus als Symbol (oben rechts) vorgesehen, welches ein Pop-up Fenster von unten aufgerufen hätte. Im Entwicklungsprozess wurde die Funktionalität umgeändert, das Design und die Hauptfunktionen wurden allerdings so gut es ging übernommen.

Die finale Kalender Komponente ermöglicht das Anzeigen der einzelnen Monate durch Klicken der Symbole “<” und “>”. Der aktuelle Tag wird mit grau markiert erkennbar gemacht. Der leicht Orange Hover Effekt verdeutlicht welchen Tag man beim Klicken auswählen würde, welcher dann Orange markiert und mit weißer Schrift angezeigt wird. Wurde ein Tag dann ausgewählt kann über das Eingabefeld der Name des Termins eingetragen werden. Bei erneutem Auswählen des Tages wird der Termin im Infofeld unten dann angezeigt und kann durch eine erneute Eingabe im Eingabefeld überschrieben werden. Dies funktioniert auch beim Wechseln zwischen den Monaten, da alle Termine mit entsprechendem Datum im Local Storage des Endgeräts des Nutzers gespeichert werden. *(siehe Kommentare im Code)*

Reflexion

An erster Stelle in der Entwicklung stand eine ausgiebige Recherche zur Erstellung eines Kalenders mit Hilfe von Typescript und Stencil. Glücklicherweise habe ich es nach gut 5 Stunden recherche und wiederholter Integration zweier verschiedener Vorlagen endlich geschafft eine davon erfolgreich in unseren Quellcode zu integrieren (https://github.com/pguso/stencil-calendar). Im nachhinein frage ich mich, warum dieser Prozess so viel Zeit für mich in anspruch genommen hat. Ich gehe davon aus, dass es generell an der intensiven Befassung mit dem Thema TypeScript und generell der Webentwicklung durch die drei Projekte: Einzelprojekt, Stencil Projekt und Projektmanagement Projekt, lag. Dieser Prozess war allerdings auch sehr verwirrend, da jedes Projekt ein wenig anders entwickelt wurde. Außerdem hat es einige Stunde für mich gedauert um den Code nach der erfolgreichen Integration so gut es ging zu verstehen und zu kommentieren.

Alles in allem hatte ich allerdings in diesem Projekt, im Vergleich zu den anderen, die kleinsten Schwierigkeiten, da man immer im direkten Austausch mit den anderen Gruppenmitgliedern stand und so die Hemmschwelle bei Problemen direkt zu fragen niedriger lag. Außerdem habe ich mir ein extra Dokument angelegt um den Überblick über alle erledigten und bevorstehenden Schritte so gut es ging zu behalten

Trotzdem war der zweite Entwicklungsschritt nicht einfach und sehr zeitaufwendig für mich. Der Code musste in der CSS noch an unser Design angepasst werden, was ca. ein bis zwei Stunden für mich in Anspruch genommen hat.

Der Anspruchsvollste aber lehrreichste Teil folgte im Anschluss. Es mussten noch Funktionen geschrieben werden, die es ermöglichten die Termine einzutragen, im Local Storage entsprechend zu speichern und bei erneutem Klicken abrufen. Die verschiedenen Storage Arten war mir bislang fremd und die Recherche und Umsetzung, sowie das abschließende Styling, waren noch einmal mindesten genauso Zeitaufwendig wie die Schritte zuvor.

(Die Reflexionen der einzelnen Projekte ist sehr schwer auseinander zu halten, da alle Projekte parallel abliefen und so die einzelnen Erkenntnisse und Erfahrungen nicht klar voneinander getrennt werden können. Eine ausführliche Reflexion zur Webentwicklung befindet sich in meiner Einzelprojekt Reflexion.)

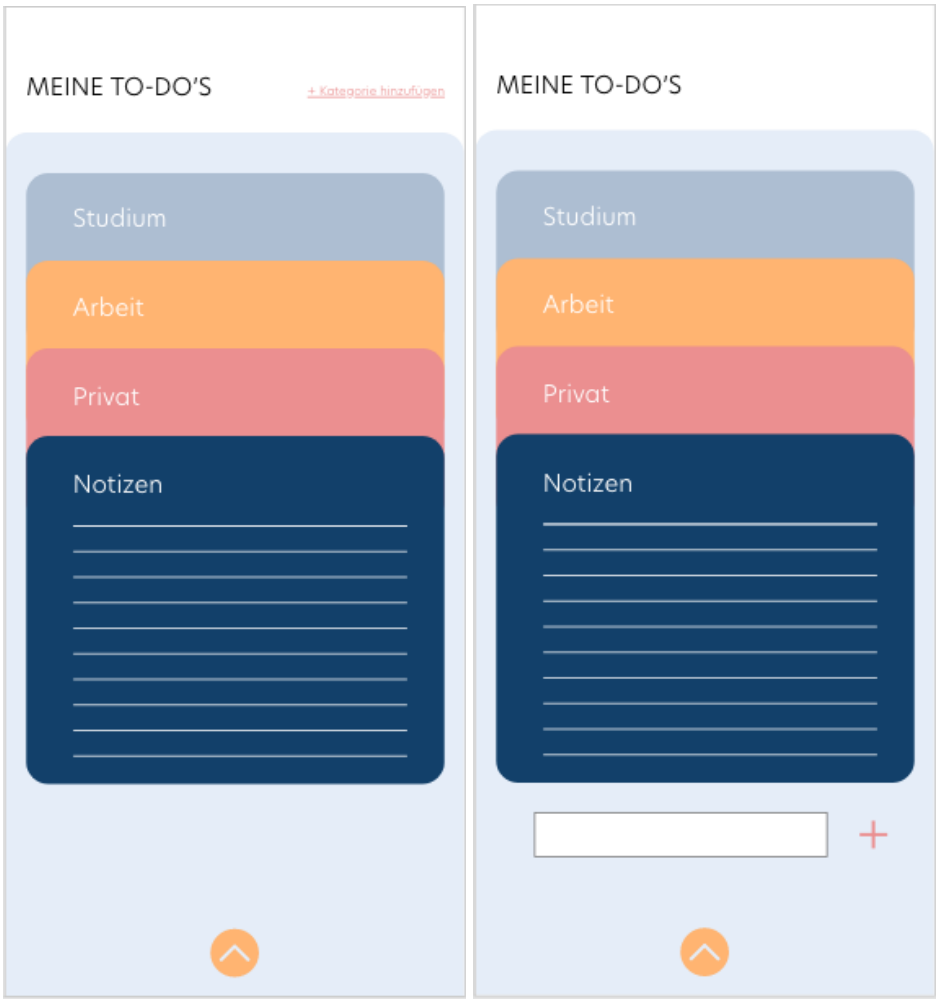
1.2 To-Do-Liste mit verschiedenen “Blättern” (Laura)

Idee

Gerade Duale Studenten können oft den Überblick über ihre To-Dos verlieren. Aus diesem Gedanken entstand die Idee eine To-Do-Liste als Komponente zu erstellen, welche aus verschiedenen “Blättern” besteht. Es gibt vier Blätter: Studium, Arbeit, Privat und Notizen.

Zuerst war mein Gedanke, dass man sowohl To-Dos/Notizen hinzufügen kann als auch neue Blätter (siehe linkes Bild). Um die Komponente für den Anfang zu vereinfachen, wurde sie entsprechend dem rechten Bild umstrukturiert. Es ist nun nicht mehr möglich einzelne “Blätter” hinzufügen. Es kann lediglich der Text für eines der “Blätter” über ein Input-Feld hinzugefügt werden. Damit der Text zur passenden Kategorie geschrieben wird, sind unter dem Input-Feld noch Radios-Buttons hinzugefügt worden.

Beim Überfahren der einzelnen Kategorien wird das jeweilige “Blatt” herausgeholt und man hat einen Überblick über die sich darauf befindlichen To-Dos/Notizen.



Reflexion

Begonnen habe ich mit dem Schreiben des HTML- und CSS-Codes, sodass das Aussehen dem Design des Prototypen gleicht.

Da sich die einzelnen “Blätter” überlappen, ist eine wichtige Funktion, dass diese beim darüber fahren, nach oben kommen, sodass der Inhalt angezeigt werden kann. Dies geschieht durch einen einfachen CSS-Hover-Effekt.

Die Problematik an dieser Komponente war, dass der eingegebene Text in die entsprechenden “Blätter” geschrieben wird. Dies soll parallel zum ausgewählten Radio-Button geschehen. Dies geschieht durch eine Funktion, die sich den Inhalt des Input-Feldes greift und mit einem “-” davor in das Blatt schreibt. Die Funktion besteht aus einer if-Abfrage, in der auch abgefragt wird, welche Box ausgewählt wurde.

Das Gruppenprojekt ist mir deutlich leichter gefallen, als das Einzelprojekt. Das Einzelprojekt habe ich früher angefangen und somit waren schon deutlich mehr Grundlagen da, als wir mit dem Gruppenprojekt angefangen haben.

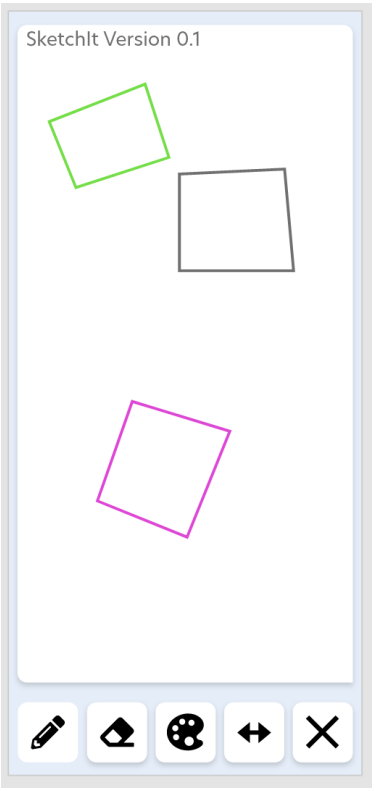
Zu Beginn haben wir die ersten Schritte gemeinsam angelegt und auch unsere Komponenten, sodass wir alle auf einem Stand waren und gleich arbeiten konnten.

Die Idee für unser Projekt stand sehr schnell, wir haben uns zu Beginn in einer MindMap Gedanken gemacht, was unsere “App” alles beinhalten könnte und uns dann für die Komponenten entschieden, die wir für besonders wichtig hielten und uns am Meisten angesprochen haben.

Die Gruppe hat an sich sehr gut funktioniert und auch die Treffen waren sehr harmonievoll und wir waren uns sehr schnell einig (siehe Gruppenreflexion).

1.3 Zeichenfläche (Dominik)

Idee



Inspiriert an der Möglichkeit in Moodle mit einfachen Werkzeugen auf einer Zeichenfläche entweder einfache Skizzen anfertigen zu können, oder auch bei Langeweile einfach ungestresst rumkritzeln zu können und seiner Fantasie freien lauf zu lassen, kam ich auf die Idee eine einfache Zeichenkomponente zu entwickeln, welche den Umfang des ON-Managers erweitert.

Ziel war es eine nahezu bildschirmfüllende Zeichenfläche zu haben, auf welcher man nach Belieben herummalen können sollte.

Am unteren Ende des Bildschirms soll es ein paar einfache aber sinnvolle Werkzeuge geben.

Mit der Auswahl des Zeichenstifts eröffnet sich einem die Möglichkeit, auf der Zeichenfläche Zeichnen zu können, sie steht im Gegensatz zum Radiergummi, diese beiden Knöpfe heben sich gegenseitig auf.

Durch einen Klick auf die Farbpalette soll die Farbe geändert werden können. Es soll nicht explizit eine Farbe gewählt werden können, sondern durch eine limitierte Auswahl durchgeschaltet werden können.

Die Auswahl der Dicke des Stiftes soll genauso funktionieren wie die Auswahl der Farbe, auch hier wird durch eine Liste von Breiten durchgeschaltet.

Durch das Betätigen des Clear-Icons soll die Zeichenfläche zurückgesetzt werden, sodass man ein neues Kunstwerk beginnen kann.

Ich werde ein Canvas verwenden, um diese Anwendung umzusetzen und außerdem externe Icons einbinden, um den Prototypen möglichst genau nachzubauen.

Reflexion

Für die Umsetzung meiner Zeichenfläche, habe ich für die Grundfunktionalität des Zeichnens eine simple Umsetzung in Javascript zuerst genauer studiert, und dann diese Schrittweise in TypeScript umgesetzt und so angepasst, dass die Zeichenfläche allgemein funktioniert, und des Weiteren so eingebaut, dass ich die von mir gedachte Funktionalität implementieren kann. Gemeint ist damit beispielsweise, dass ich statt fixer Werte für meinen Pinsel diesen komplett mit variablen verziert habe, welche ich anschließend durch die dafür vorgesehen Bedienflächen verändern kann. Außerdem wird jede Veränderung der Zeichenfläche im Localstorage gespeichert. Das hat erstaunlicherweise ohne größere Probleme funktioniert.

Bei der Auswahl des Zeichenstifts, werden die aktuell gesetzte Farbe, und Stiftbreite verwendet. Die Auswahl des Radiergummis ist dadurch gelöst, dass die Stiftbreite gleich bleibt, die Farbe jedoch lediglich zu weiß geändert wird. Es wird sich vom Browser, durch den Localstorage gemerkt, ob der Stift oder der Radiergummi zuletzt gewählt war.

Die wählbaren Farben, stammen aus einer vorbestimmten Farbpalette, und die aktuell gewählte Farbe wird im Localstorage gespeichert, sodass wenn man die Seite wieder besucht, die zuvor konfigurierten Einstellungen behält, und dort weitermachen kann, wo man aufgehört hat. Ich habe mich dafür entschieden die Auswahl zu limitieren, um nicht unnötig viele und komplexe Bedienflächen zu haben, sondern die Anwendung leicht verständlich und Clean zu halten.

Ähnlich wie bei der Umsetzung der Farbe, ist auch die Stiftbreite limitiert, und wird ebenfalls im Localstorage gespeichert.

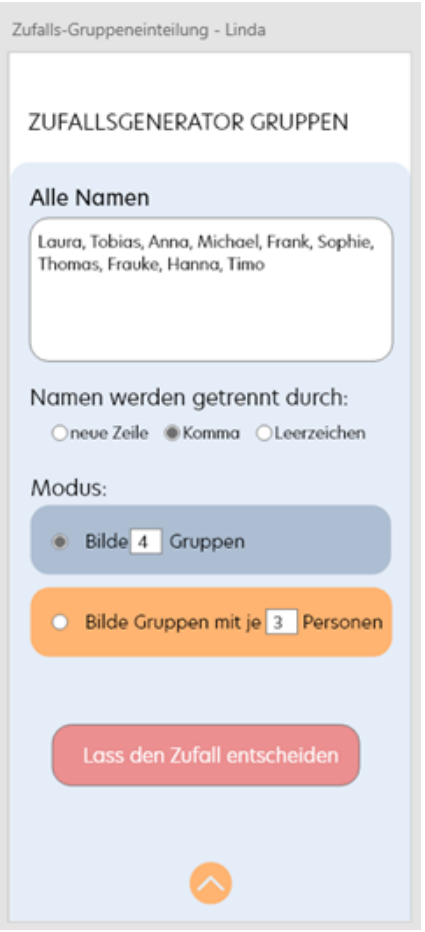
Clear reinigt die Zeichenfläche von allem gemalten, und aktualisiert auch die Version des Canvas im Localstorage.

Zuletzt habe ich noch Text hinzugefügt, welcher Auskunft über die aktuell gesetzten Parameter des Canvas gibt. Die erleichtert die Bedienung und das Verständnis der Komponente, und gibt dem Nutzer ein Feedback, dass etwas passiert, wenn die Knöpfe bedient werden.

Die Icons auf den Knöpfen stammen von FontAwesome. Das einbringen in die Komponente sorgte anfangs für Probleme, und wurde Notgedrungen über Slots umgesetzt.

1.4 Zufallsgenerator Gruppen (Linda)

Idee



Im Alltag eines Online-Medianer sind sie fast täglich gefragt, die Gruppeneinteilungen. Der Vorgang zur Gruppenfindung nimmt in vielen Vorlesungen meistens um die 10 Minuten in Anspruch. Die schnelle Lösung: der online Zufallsgenerator für Gruppen! Einen Zufallsgenerator zur zufälligen Gruppenerstellung durch welchen beliebig viele Gruppen erstellt werden können. Egal ob Einteilung in beliebig viele Gruppen oder beliebig viele Personen pro Gruppe, der Zufallsgenerator erleichtert die Arbeit enorm.

Hierzu können die einzelnen Namen der Studenten mit drei verschiedenen Trennmethoden eingegeben werden, diese dienen dazu, dass schon vorgefertigte Listen schnell mit der passenden Trennmethode kombiniert werden können. Um dem Nutzer zu zeigen, wie er seine Namen eintragen muss, ist unter jedem Trennmodus ein unterschiedlicher Placeholder zur Vorschau versteckt. Der Trennmodus Leerzeichen wünscht die Namen nacheinander eingetragen und mit einem Leerzeichen getrennt, “neue Zeile” wünscht jeden Namen mit einem anschließenden enter getrennt, sodass alle Namen untereinander stehen und “Komma” wünscht die Namen durch ein Komma mit anschließendem Leerzeichen getrennt.

Es gibt zur Gruppenerstellung zwei mögliche Modi. Modus 1 erstellt aus der Namensliste jeweils die gewählte Anzahl an Gruppen, welche zuvor eingetragen werden kann. Ist die Liste ungerade, fällt die letzte Gruppe kleiner aus. Modus 2 erstellt aus der Namensliste so viele Gruppen wie nötig sind um die Personenanzahl, welche zuor eingetragen wurde, der Gruppen einzuhalten. Ist die Anzahl der Namen ungerade, wird die letzte erstellte Gruppe mit weniger Personen ausgegeben.

Sind die Namen eingetragen, ein Trennmodus und ein Modus ausgewählt, erhält man mit dem Klick auf den Button “Lass den Zufall entscheiden” einen Alert mit den zufällig eingeteilten Gruppen. Unter dem Button befindet sich ein Pfeil, welcher bei Klick darauf den Footer erscheinen lässt.

Reflexion

Nachdem das HTML und CSS Gerüst fertig war, ging es darum die Logik für den Zufallsgenerator zu schreiben. Um herauszufinden welchen Trennmodus und welchen Modi der Nutzer anhand der Radiobuttons ausgewählt hat wurde die Funktion buildGroups erstellt. Diese prüft welche Radiobuttons ausgewählt wurden.

Schwieriger wurde es bei der Logik der zwei verschiedenen Modi. In meinem Fall habe ich beim ersten Modi begonnen und überlegt was brauche ich alles, um später das gewünschte Ziel zu erreichen. Um später zufällige Gruppen zu erhalten, mussten die Namen zuerst durchgemischt werden. Dies passiert in der groupBuilderFunktion gleich am Anfang. Die gemischten Namen werden in der variable “shuffledList” gespeichert. Nun war es relativ leicht die Anzahl der Gruppen zu bestimmen, denn hierzu musste ja nur der Wert des Input Feld aus der HTML abgefragt werden. Eine Schwierigkeit bestand darin, dass es ja auch möglich ist, dass die Namen nicht ohne Restwert in die angegeben Gruppenanzahl passten. Nach längerer Überlegung und Hilfe durch Dominik kamen wir gemeinsam auf die Idee dies mit einer If Bedingung und einem Modulo Operator zu lösen.

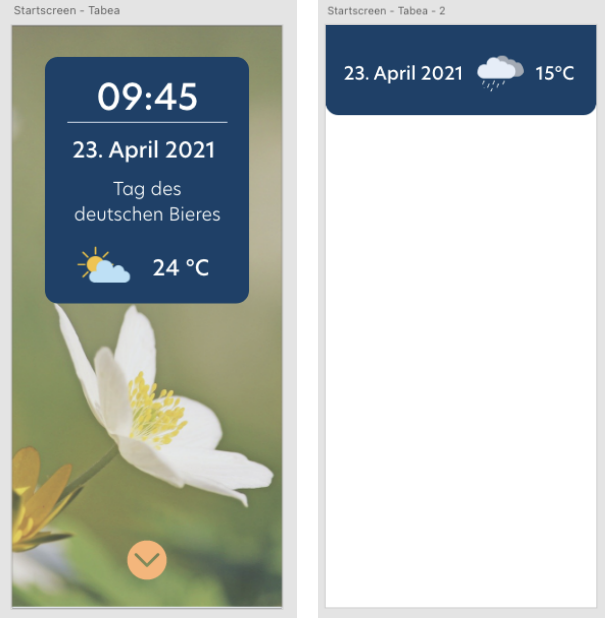
Modus zwei brachte wiederum andere Schwierigkeiten mit sich, denn die Gruppenanzahl stimmte hierbei nicht mit dem Input Feld überein. Um die Gruppenanzahl hier zu erhalten wurde die Variable “sumGroups” eingeführt. Diese teilt die Anzahl der eingegebenen Namen durch den Wert in dem Input Feld. Leider gab es hier das Problem, dass die letzte Gruppe (wenn es einen Restwert gab) immer die Anzahl mit dem Wort undefined gefüllt hat. Dieses Problem habe ich im Team angesprochen und wir haben als gemeinsame Lösung eine weitere if-Bedingung ergänzt, welche der letzten Gruppe die restlichen Mitglieder zuweist.

Das größte Problem hatte ich bei der Textarea für die Namen, hier hatte ich mir gewünscht, dass bei Auswahl der verschiedenen Trennmöglichkeiten der Placeholder verändert werden kann. Um zur Lösung zu gelangen musste für jede Möglichkeit eine Funktion geschrieben werden, welche dann ausgelöst wird, wenn die Radiobuttons für die Trennmodi verändert werden.

Das Gruppenprojekt war für mich einfacher als das Einzelprojekt, da ich die Unterstützung der Gruppe hatte, welche sich in unserem Fall gegenseitig geholfen und unterstützt hat. Ebenso konnte ich mir zwischendurch die Komponenten der anderen anschauen und daraus neue Ideen und Anregungen sammeln.

1.5 Start-Screen (Tabea)

Idee



Ziel war es einen reduziert gestalteten Start-Screen zu erstellen (ursprüngliches Design s. Abb.), den der User auch während des Lernens geöffnet haben kann ohne abzulenken.

Der User soll die Uhrzeit im Blick behalten können. Damit er das Datum direkt auf seine Aufschriebe übertragen kann soll auch das aktuelle Datum eingeblendet sein.

Zur Auflockerung und um Gespräche mit Kommilitonen anzuregen wird an entsprechenden Tagen der jeweilige "Welt-Tag" (z.B. Welttag des Designs) angezeigt.

Außerdem hat der Student die Möglichkeit das aktuelle Wetter, also ob es regnet, schneit, bewölkt oder klar ist, und die aktuelle Temperatur in Mosbach einzusehen um sich entsprechend wettertauglich anziehen zu können. Das Wetter wird alle 15 Minuten aktualisiert, so dass der User immer auf dem neusten Stand ist. Das Wetter wird zusätzlich durch den sich abhängig vom Wetter anpassenden Hintergrund greifbar gemacht.

Wichtige Informationen, wie das Wetter und das Datum, sollen auch während der Nutzung der App sichtbar bleiben. Deshalb verkleinert sich der Start-Screen beim scrollen auf einen Balken auf dem diese Informationen sichtbar bleiben und der am oberen Bildschirmrand stehen bleibt. Dadurch kann der User diese Informationen auch bei der Nutzung anderer Komponenten der App sehen.

Es wurde ein animierter Button eingebaut, der den User darauf hinweist, dass er scrollen muss um auf die weiteren Inhalte der App zu gelangen.

Reflexion

Uhrzeit & Datum

Die Uhrzeit wurde über das Interface Date eingebunden.

Für die Uhrzeit wurde mit dem Interface Date die Stunde und Minute ausgegeben. Eine besondere Herausforderung dabei war, dass bei Stunden und Minuten, die kleiner sind als 10, das übliche Format mit vorgesetzter Null angezeigt wird. Dazu habe ich mich an "https://gist.github.com/Konstantinos-infogeek/8f89e07139bc3dacea371e3cf2bc556c" (Stand: 28.04.21) orientiert.

Das Datum wurde auch mit dem Interface Date abgerufen und mit Hilfe der toLocalDateString-Methode und dem Interface DateTimeFormatOptions in das in Deutschland übliche Format mit ausgeschriebenem Monat gebracht. Dazu habe ich mich an https://stackoverflow.com/questions/1531093/how-do-i-get-the-current-date-in-javascript (Stand: 28.04.21) orientiert.

"Welt-Tag"

Nachdem ich einen halbwegs offiziellen Begriff für diese Art an besonderen Tagen gefunden habe bin ich leider bei der Suche nach einer geeigneten API nicht fündig geworden. Ich habe also anhand einer Auswahl aus http://webtimum-online.de/kuriose-feiertage/#Witzige_kuriose_Feiertage (Stand: 28.04.21) eine Json-Datei angelegt und die Informationen aus ihr gezogen.

Hier war eine Herausforderung die Daten aus der Json-Datei auszulesen und nutzbar zu machen und die Tage und Monate mit dem aktuellen Tag und Monat zu vergleichen. Nachdem ich etwas Nachhilfe bei meinem Bruder zum Thema Promise und anonymen Funktionen in Anspruch genommen habe und mich nochmal mit den Lifecycle-Methoden auseinandergesetzt habe, habe ich es aber gelöst bekommen.

Wetter-Daten

Die Wetterdaten stammen von https://openweathermap.org/ . Mit der kostenfreien Version ist die Anzahl der Aufrufe in einem bestimmten Zeitraum begrenzt. Nachdem ich die Wetterdaten und auch den Scroll-Effekt eingebunden hatte und wir parallel am Code gearbeitet haben kam es zu Problemen. Wir haben festgestellt, dass vermutlich vor allem durch den Scroll-Effekt, dieses Limit schnell erreicht wird.

Um die Begrenzung der API-Aufrufe zu begrenzen habe ich erneut Hilfe in Anspruch nehmen müssen und mir Tipps und Hints geben lassen, dass ich mich erneut

verstärkt mit den Lifecycle-Methoden und Decoratorn auseinander setzen muss, bis ich schließlich eine Lösung gefunden habe.

Wechselnder Hintergrund

Bei dem wechselnden Hintergrund habe ich für verschiedene Wetterlagen, wie klar, bewölkt, Regen, Schnee, Gewitter und windig, Bilder von https://pixabay.com und https://www.freepik.com gesucht und entsprechend bearbeitet, so dass sie farblich zum Hintergrund passen. Über eine if-else-Abfrage wird festgestellt welches Wetter aktuell ist und das entsprechende Bild dargestellt.

Scroll-Effekt

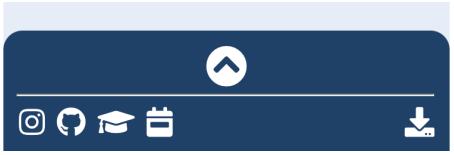
Beim Scroll-Effekt sollen sich die Elemente abhängig davon ob der User in y-Richtung scrollt bewegen. Dazu musste ich abfragen ob gescrollt wurde und den Wert speichern. Da zu diesem Zeitpunkt bereits gerendert wurde musste ich mich daran erinnern, dass mit dem State-Decorator dafür gesorgt werden kann, dass bei einer Änderung neu gerendert wird. Darüber hinaus hat es mich einige Zeit gekostet auf die Idee zu kommen die Inhalte der App, die gerendert werden sollen, in eine Methode auszulagern. So konnte ich abfragen welchen Y-Offset ich habe und abhängig davon die Elemente rendern lassen. Leider war so kein flüssiger Übergang vorhanden, sondern der Wechsel ist abrupt. Zum einen musste ich einstellen, dass die Änderung nicht zu empfindlich reagiert, sobald der User sein Mausrad oder den Bildschirm berührt, indem der Effekt erst ab einem Y-Offset von 20 beginnt.

Wie ich einen flüssigen Übergang schaffen kann war mir erst nicht klar. Nachdem ich den Hinweis erhalten habe, dass ich das mit Hilfe von CSS-Animationen lösen kann habe ich es damit umgesetzt. Da sich die Größe von Elementen durch Padding verändert war es nicht schwierig passende Animationen anzulegen. Der Übergang findet auch nur in eine Richtung statt um eine Lösung für beide Richtungen zu suchen, ohne dass die Animation bereits beim ersten Aufruf oder dem Neuladen der Seite stattfindet, hat mir leider die Zeit nicht gereicht.

1.6 Footer (Tobias)

Idee

In keiner Website sollte er fehlen - der Footer. Ziel war es einen intuitiven Footer zu entwickeln. Dieser Footer soll auf einem Blick die relevantesten Links für Online-Medianer bereitstellen. Von den Vorlesungsplänen, zu dem STUV-Instagramkanal, bishin zu unserem GitHub-Repository, ist alles dabei. Ebenso befindet sich neben den ganzen Social-Media-Icons, ein Download-Button, welcher es ermöglicht unsere Dokumentation herunterzuladen. Anfangs war es geplant einen Footer zu entwickeln, welcher sich statisch, ganz unten auf der Seite befindet. Mit einem “Back-to-top”-Button sollte man dann wieder nach ganz oben gelangen.



Da wir jedoch in einem Stencil-Projekt arbeiten, und Stencil im Grundlegenden davon lebt, dass man die Komponenten einer Seite, mehrmals aufgreifen kann, wurde aus dem statischen Footer ein dynamischer Footer, welcher auf jeder Komponente erreichbar ist. Da der Footer mit seinem all seinem Inhalt zu viel Platz wegnehmen würde, wurde eine dynamische Funktion eingebaut, welche es ermöglicht den Footer ein- und auszuklappen. Der Normalzustand des Footers ist also der eingeklappte Zustand. Wenn man weitere Infos möchte, klappt man diesen aus und erreicht somit seinen vergrößerten Zustand.

Begonnen habe ich damit, ein HTML/CSS-Grundgerüst zu erstellen, welches den Vorstellungen unseres Prototypen entsprach. Um passende Icons im gleichen Style einbauen zu können, habe ich auf das Toolkit “FontAwesome” zurückgegriffen. Als das Styling soweit fertig war, ging es darum, die Icons und Buttons mit passenden Links auszustatten. Die länglichen Buttons wurden mit einer “onClick-Funktion” eingebunden. Die Fontawesome-Icons musste ich über die HTML-Datei mit Links ausstatten. Am Ende musste der Footer natürlich auch noch dynamisch werden. Dieses Problem wurde dann größtenteils mit einer “Toggle-Funktion” gelöst, welche auf zwei verschiedene CSS-Klassen zugreift. Wenn der Footer zugeklappt ist, wird also die Klasse “hiddenContent” ausgeführt. Am Ende musste ich noch dafür Sorgen, dass der Footer auf jeder Komponente unten fixiert ist. Als dies einwandfrei funktionierte, war mein Footer vollendet.

Reflexion

Die größten Probleme hatte ich tatsächlich mit dem Googlen. Ich hatte das Gefühl, dass man schon über das Thema “Typescript” relativ wenig fand. Aber als es dann spezifischer wurde, und man auch noch Rücksicht auf Stencil nehmen musste, gestaltete sich das Finden von passenden Lösungen weitaus schwieriger. Ebenso gestaltete sich das ein- und ausblenden des Footers nicht gerade als einfach. Ein weiteres Problem war, dass man in der tsx-Datei Fontawesome nicht richtig einbinden konnte. Deshalb mussten Dominik und ich unsere Icons eher unschön über die globale “index.html” einbinden.

Trotzdessen, dass ich selbst nicht der geborene Entwickler bin und ich mich auch später eher in einer anderen beruflichen Richtung sehe, hat mir das Projekt sehr viel Spaß gemacht. Mir gefällt die Idee von Stencil mit seinen Komponenten sehr. Wir arbeiteten in der Gruppe so, wie man sich das auch vorstellt - strukturiert und konzentriert. Wenn jemand eine Frage hatte, war immer direkt jemand zur Stelle, der helfen konnte. Dadurch, dass ich im Vergleich zu den anderen eine eher kleine Komponente hatte, konnte ich die Verbleibende Zeit gut nutzen und die anderen bei der Entwicklung ihrer Komponenten unterstützen.

Zusammenfassend war das Projekt für uns alle ein voller Erfolg. Ich denke wir können sehr stolz auf unser finales Produkt sein. Manche Funktionen, wie die Gruppeneinteilung, werden wir vielleicht im Laufe unseres Studiums tatsächlich noch gebrauchen können.

2. Reflexion Gruppenarbeit

Die Gruppenarbeit hat uns allen Spaß gemacht und uns vor allem in unseren Programmierfähigkeiten bereichert. Alle Teammitglieder haben ihr bestes geleistet um die eigene Komponente zu erstellen und anzupassen. Dazu muss gesagt werden, dass wir des öfteren gemeinsame Meetings hatten und zusammen an einzelnen Problemen der Komponenten weiterzuarbeiten. Über die gesamte Projektzeit bewies sich Dominik als sehr hilfsbereit. Egal welche Probleme die einzelnen Teammitglieder mit ihren Komponenten hatten, hatte er immer einen Rat parat und half das Problem zu lösen.

Insgesamt waren wir ein sehr gutes Team die alle am gleichen Strang gezogen haben und auch die schwierigen Aufgaben unserer App mit Humor und Spaß bewältigt haben. Vielen haben die Ansätze des Gruppen Projektes auch für das Einzelprojekt geholfen, was sich für alle positiv ausgewirkt hat.

Großer Dank geht nochmals an unseren Programmier-Leader Dominik, der immer für alle da war und somit auch in der Gruppenreflexion und vor allem bei der Benotung besonders hervorgehoben werden sollte.