FHS St.Gallen
Hochschule
für Angewandte Wissenschaften

SWI1 Abschlussbericht Amar Redzepi

Quellennachweis Titelblatt

Webseitenentwicklung in Deutschland – rollenspiel-welten.de. (ohne Datum). [Foto]. Abgerufen von http://rollenspiel-welten.de/webseitenentwicklung-in-deutschland/

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Mockups

In diesem Kapitel werden die erstellten Mockups für den Werkzeugkasten näher beschrieben. Dabei sind vier verschiedene Mockups erstellt, einmal für die Web Page und drei Mockups für die Mobile Version der Webseite. Die Mockups sind mit Wireframe.cc erstellt. Diese sind Vorstellungen, wie die Webseite ausschauen sollte, sie kann aber vom Endprodukt abweichen.

1.1 Web Page

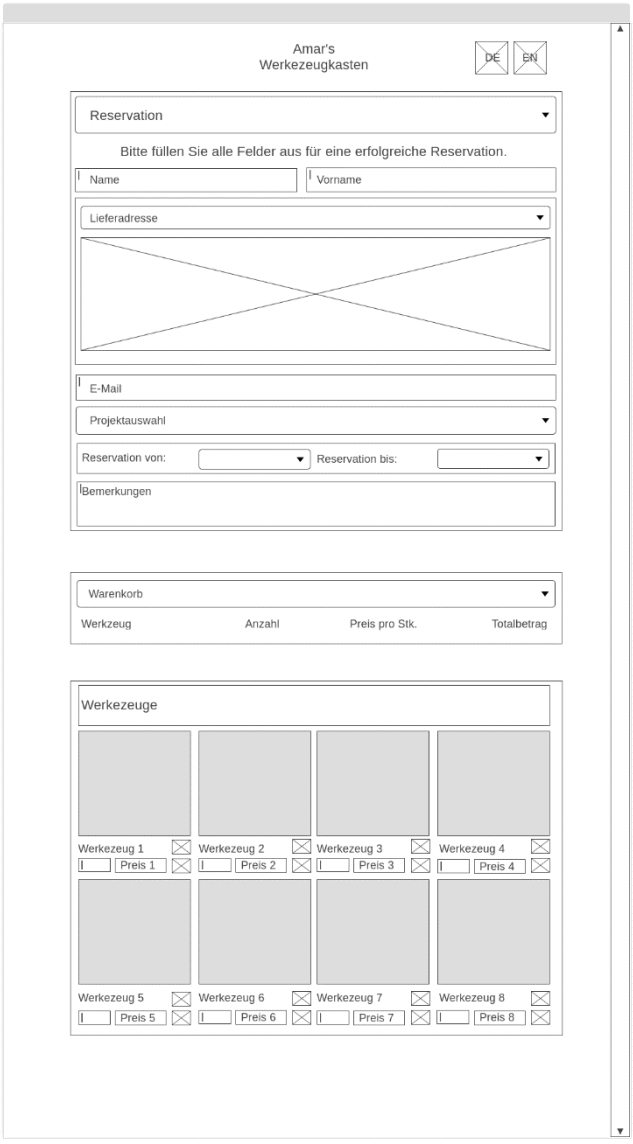

Abbildung 1: Mockup Webseite
Quelle: eigene Darstellung.

Die Webseite startet mit dem Titel, wobei rechts die Optionen für «Deutsch» und «Englisch» Benennungen der Werkzeuge zur Verfügung stehen. Zuerst ist die Reservation aufgezeigt, worin alle reservationsrelevanten Informationen eingetragen werden. Es ist ausserdem eine Karte aufgezeigt, wo die Koordinaten des Standorts angezeigt werden. Reservation von – bis wird mittels Kalender angezeigt, wobei der bestimmte Tag ausgewählt werden kann.

Anschliessend ist eine Übersicht der reservierten Artikel im «Warenkorb» ersichtlich. Dort werden nach Auswahl der Werkzeuge in der Werkzeugliste die Werkzeuge, die Anzahl, der Preis pro Stück und der Totalbetrag dargestellt.

Die Werkzeugliste stellt die Werkzeuge mit Ihrer Bezeichnung, dem Preis, den Button fürs Reservieren sowie einem Button mit Zusatzinformationen dar.

1.2 Mobile-App Reservation

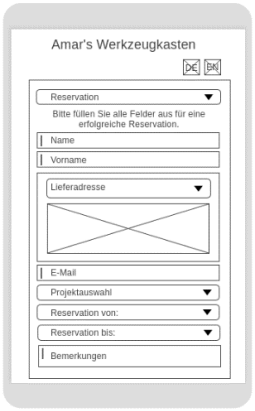

Abbildung 2: Mockup "Reservation"
Quelle: eigene Darstellung.

Die Reservation der Werkzeuge auf dem Mobile wird auf einer Übersicht dargestellt, sodass alle Informationen ohne Scrolling erfasst werden können. Die Reservation-Bar ist auf und zu klappbar. Wenn Informationen vergessen werden, wird der User benachrichtigt, dass Eingaben fehlen oder fehlerhaft sind.

1.3 Mobile-App Werkzeugliste

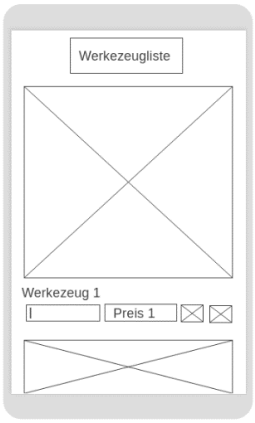

Abbildung 3: Mockup "Werkzeugliste"
Quelle: eigene Darstellung.

Die Werkzeuge in der Mobile App werden einzeln mit den gleichen Informationen wie auf der Webseite dargestellt. Das Design ist responsiv, weshalb man hier ein Werkzeug auf dem Bildschirm sehen kann und nicht mehrere Werkzeuge.

1.4 Mobile-App Warenkorb

Der Warenkorb stellt wie auf der Webseite die Artikel in einer Tabelle dar, welche horizontal scrollbar ist. Dies vor allem weil die Tabelle sonst auf der Mobile App keinen Platz hat. Der Warenkorb ist ebenfalls auf und zu klappbar.

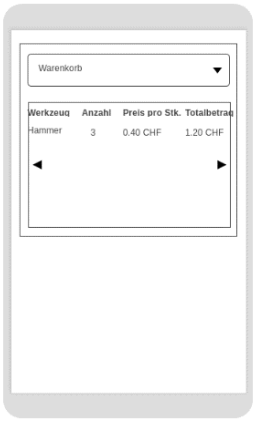

Abbildung 4: Mockup "Warenkorb"
Quelle: eigene Darstellung.

2 Umsetzung

2.1 Auftrag 1: Grundgerüst

Die Erstellung des Grundgerüsts verlief relativ problemlos. Dabei habe ich das index.html, script.js und styling.css File erstellt sowie Unterverzeichnisse für die Bilder gemacht. Für das responsive Design habe ich mich von Anfang an entschieden Bootstrap anzuwenden, damit das responsive Design auch eingehalten wird. Bilder habe ich für die Button der Sprachen und für den Fehlerfall (Eingabe 0) im Kostenrechner (GIF) eingebaut. Ich habe in der Webseite ein zentrales Formular (meinformular) eingebaut um alle relevanten Elemente erfolgreich mittels POST Methode an den Server zu senden. Dabei habe ich den Reservierungsantrag und den Kostenrechner in das Formular eingeschlossen, um alle Informationen gemäss Vorgaben (firstname, lastname etc.) sowie die Werkzeuge, welche bestellt werden zu versenden. Buttons sind für die Sprachänderung und für das Absenden des Reservierungsantrages eingebaut werden.

2.2 Auftrag 2: Dynamische Elemente und Wahlaufgaben

2.2.1 Pflichtaufgaben

Für das dynamische Generieren der Werkzeugliste und des Kostenrechners habe ich zunächst die notwendigen Get JSON Statements programmiert und mir danach überlegt, in welcher Art und Weise ich diese im HTML darstellen möchte. Nach einiger Recherche habe ich verschiedene Möglichkeiten aufgefunden, unter anderem Columns, Tabellen und Cards. Der Entscheid traf die Cards, wobei ich für jedes Werkzeug dynamisch eine Card generiert habe. Jede Card beinhaltet die Bezeichnung (Label), den Bestand (stock), den Preis (prize) und ein individuelles Inputfeld, jedoch ohne Button (welcher ursprünglich geplant war). Wenn keine Werkzeuge auf Lager sind, dann wird das Inputfeld «disabled» und farblich gekennzeichnet.

Für die Optionen der Projektauswahl habe ich ebenfalls einen Get JSON Funktion programmiert, welche die Optionen dynamisch im HTML der Reservation (projekte) einspeist.

Das Einfügen einer Zahl im Inputfeld generiert im Kostenrechner dynamisch zwei separate Ansichten. Eine Ansicht beinhaltet alle Preise in CHF, die andere in EUR. Die Europreise sind im Prinzip die CHF Beträge multipliziert mit dem Kurs von exchangerate.api. Der Eurokurs ist ebenfalls mit einer Get JSON Funktion abgerufen worden. Zusätzlich werden die Zeitangaben aus der Reservation übernommen. «Datum von» und «Datum bis» werden von einander subtrahiert, auf ein Tagesformat gebracht und mit dem Preis pro Tag multipliziert. Hier liegt das Problem, dass wenn keine Zeitangaben angegeben wurden der Text «Zeitraumangabe fehlt.» steht. Die Zeitangaben werden nur dann aktualisiert, wenn ein Inputfeld im Werkzeugkasten

geändert wird, was nicht sehr praktisch ausgefallen ist. Damit bekommt man das Gesamttotal. Der Kostenrechner wird mit jeder neuen Eingabe komplett gelöscht und alle Elemente durchgegangen. Die Elemente, welche einen Input haben werden in den Kostenrechner überführt. Erst wenn etwas im Kostenrechner drinnen ist, kann der Reservationsantrag versendet werden. Dies habe ich so gelöst, indem ich eine Variable «warenkrbleer» definiert habe, welche den Status jeweils anzeigt und aufgrund dieses boolschen Wertes beurteilt, ob etwas im Warenkorb ist oder nicht.

Im Kostenrechner werden wie bereits erwähnt die Ansichten dynamisch generiert. Wenn jedoch eine «0» im Inputfeld eingegeben wird, kommt eine Fehlermeldung mit einem alert, welcher darauf hinweist, dass der Input korrigiert werden muss, sonst wird der Kostenrechner nicht richtig angezeigt. Dabei wird das «wrong.gif» eingeblendet und der Antrag kann nicht versendet werden. Somit wird verhindert, dass 0 Werkzeuge reserviert werden.

2.2.2 Wahlaufgaben

Für die grafische Auswahl des Datums habe ich den Datepicker von Bootstrap eingebaut, welcher einfach und schnell funktioniert hat. Umfangreich war jedoch die korrekte Verarbeitung der Eingaben für den Kostenrechner (wie vorhin bereits beschrieben).

Die Sprachauswahl habe ich mit zwei Buttons gelöst, welche unterhalb vom Seitentitel dargestellt werden. Wenn der Button «Deutsch» gedrückt wird, dann werden alle deutschen Label in der Werkzeugliste angezeigt, und die englischen Label ausgeblendet. Genau das Gegenteil passiert beim Button «English». An dieser Stelle ist anzumerken, dass ich beim Erzeugen der Werkzeuge in der Werkzeugliste ein zusätzliches Element eingebaut habe, welches die Label in Englisch beinhaltet. Diese englischen Labels sind nach Ihrer Erzeugung nicht sichtbar, sondern nur die deutschen Labels. Ich habe im CSS die englischen Label auf «display: none» und «visibility: hidden» eingestellt und mit Hilfe einer eigenen Funktion (zeigeElement();) diese Eigenschaften auf «visibility = "visible";» und «display = "block";» geändert. Erst wenn die Buttons geklickt werden, wird mit Hilfe der jQuery Funktionen show(), hide() und meiner Funktion zeigeElement() die Elemente angezeigt oder versteckt und entsprechend die richtige Sprache angezeigt.

2.3 Auftrag 3: Daten versenden und Wahlaufgaben

2.3.1 Pflichtaufgaben

Die Formularvalidierung habe ich so gelöst, indem jedes relevante Inputfeld «required» ist (sprich im HTML Code des Inputfeldes so definiert). Wenn ein Input fehlt, wird dies dem User mitgeteilt und er kann entsprechend die Felder ergänzen. Dies bietet Bootstrap an und mithilfe des «input type» kann die richtige Formatierung geprüft werden. Bei falscher Formatierung habe

ich ein «invalid-feedback» ausgegeben, welcher dem User noch vor dem Absenden mitteilt, ob ein Input fehlt oder falsch erfasst ist. Zusätzlich habe ich bei den beiden Inputfelder für «Name» und «Vorname» eine Regex Formel eingebaut, damit nur Gross- und Kleinbuchstaben akzeptiert werden. Das Problem dabei war, dass der «input type=text» auch Zahlen erlaubt und ohne der Regex Bedingung könnten theoretisch auch Zahlen als Name und Vorname versendet werden. Dies wird damit verhindert.

Mit dem Button «Reservieren» wird der «submit» für die Übermittlung der Anfrage ausgelöst. Zunächst wird mit Hilfe der Variable «warenkrbleer» geprüft, ob ein Werkzeug im Kostenrechner eingefügt wurde. Falls dies nicht der Fall ist, dann wird der submit unterbrochen und eine Meldung «Der Warenkorb ist leer!» im alert ausgegeben. Falls der Kostenrechner Werkezeuge beinhaltet, dann werden die reservierungsrelevanten Daten, welche im Formular «meinformular» einen «name» vergeben haben mit der Funktion serialize() serialisiert. Dies betrifft folgende Daten: firstname, lastname, email, projid, comment, id (Werkzeug) und quantity (Input Werkzeug).

```
warenkorb += '<input hidden name="' + $(this).attr("name") + '" value="' + $(this).val() + '">'
```

Abbildung 5: Ausschnitt script.js (ID und Anzahl)
Quelle: eigene Darstellung.

Der Kostenrechner fügt bei jedem Werkzeug diesen «hidden input» ein, welcher im Kostenrechner nicht angezeigt wird, aber dann im serialize() verwendet wird und mitgesendet wird (ID und Stückzahl der Werkzeuge).

Wenn dann mit der Ajax Funktion (POST Methode) die Daten erfolgreich versendet wurden, dann kommt der alert("Resultat: " + data.message), welcher die Erfolgsnachricht anzeigt (Hier: «Werkzeuge erfolgreich reserviert»). Falls ein Fehler aufgetaucht ist, wird mithilfe vom alert("Fehler: " + xhr.responseJSON.message) angezeigt, worum es sich beim Fehler handelt und was der User tun sollte.

Nachdem der Antrag erfolgreich reserviert ist, werden Name, Vorname und Email im LocalStorage gespeichert. Somit werden diese beim nächsten Aufruf der Webseite vorausgefüllt angezeigt.

2.3.2 Wahlaufgaben

Für eine geeignete Schrift habe ich fonts.google.com verwendet. Diese kam in den Titel des Kostenrechner sowie die Beschriftungen der Werkzeuge in der Werkzeugliste zu tragen. Es betrifft die Schrift «font-family: 'Arimo'» und «Roboto Slab', serif».

Mithilfe der Anleitung von Leaflet konnte ich die Karte erfolgreich erstellen. Sie zeigt von Anfang an die Schweiz im Überblick (Koordinaten wurden dabei manuell durch mich voreingestellt). Sobald die Reservation erfolgreich ist, geht die Karte mithilfe der flyTo() Funktion auf die von

der «data» zur Verfügung gestellten Koordinaten (data.pickup.latitude, data.pickup.longitude) und zeigt mit einem Popup an, wo die Ware abgeholt werden kann.

Eine CSS-Animation habe ich für den Seitentitel und für alle drei Collapse (Reservation, Kostenrechner, Werkzeugliste) eingebaut. Diese habe ich von der Webseite https://danenden.github.io/animate.css/. Die Animation ist die «fadeInUp» mit einem Delay von 0.5 Sekunden.

3 Herausforderungen

Die beiden grössten Herausforderungen für mich waren das dynamische Generieren der Werkzeugliste und die spätere Logik des Kostenrechners. Da ich zunächst mit dem dynamischen Generieren der Codes angefangen habe, hatte ich Probleme den Code richtig und sinnvoll zu schreiben. Mit der Unterstützung vom Internet und Klassenkameraden habe ich das auch in den Griff bekommen.

Wie man es im Programmiercode sicherlich erkennen kann, ist die Funktion kosten(), welche für das dynamische Generieren des Kostenrechners bestimmt ist komplex aufgebaut. Ich habe mit dieser Funktion versucht alle Fehlerquellen im Kostenrechner zu finden und sozusagen zu «entschärfen». Das hat sehr lange gedauert bis der Kostenrechner so geworden ist wie ich ihn haben wollte. Manchmal gab es auch Konflikte zwischen den etablierten Lösungen. Mein Ziel war es unter anderem keine Eingaben mit 0 zu erlauben, wenn keine Werkzeuge im Kostenrechner drinnen sind kein Reservieren möglich machen, die Sprache (nur Labels) im Kostenrechner auch anpassen wenn oben mit dem Button die Sprache geändert wird (Funktioniert nur wenn eine Änderung in einem Inputfeld gemacht wurde, nachdem die Sprache mittels Button geändert wurde), korrekte Rückmeldungen zurückzugeben, wenn etwas nicht sauber gemacht wurde etc. All diese Dinge haben enorm viel Zeit beansprucht.

Das kann daran liegen, dass ich meinen Kostenrechner mit Hilfe der Inputfelder gemacht habe und nicht mit der Logik von Buttons, welche Werkzeuge einzeln in den Kostenrechner einfügen. Ich fürchte aber eher, dass die Funktion kosten() aufgrund ihrer steigenden Grösse an Komplexität gewonnen hat und besser in Einzelteile zerlegt werden sollte. Für mich als Entwickler ist die Funktion sehr gut nachvollziehbar, aber jemand der sich den Code das erste Mal anschaut hat eventuell Probleme, die Funktion vollumfänglich zu verstehen.

Teilweise versuchte ich ein Problem auf eine gewisse Art und Weise selbst zu lösen, wobei es bereits Lösungen gibt, welche bspw. von Bootstrap angeboten werden. Im Code habe ich einen Versuch auskommentiert, wo ich mithilfe von Regex Formeln das Format für die Inputfelder in der Reservation festlegen wollte. Später habe ich dann gemerkt dass das überhaupt nicht nötig ist und die «input type» einem die ganze Arbeit wegnehmen (bis auf den «input type=text»).