Das ist ein Bingo!

Dynamische Singlepage-Website - Web-Engineering 1 HSZG

Maximilian Lehmann

Inhaltsverzeichnis

1 Einleitung

Im Folgendem werde ich mein Projekt “Das ist ein Bingo!“ näher erläutern. In der Modulvorlesung Web-Engineering 1 wurden wir vor die Aufgabe gestellt, eine dynamische Singlepage-Application für den Webbrowser unseres eigenen Designs zu entwickeln. Ich habe mich für die Umsetzung eines Spiels entschieden, das sogenannte Wort-Bingo.

Wort-Bingo bezeichnet ein Spiel, bei dem jeder Spieler vor sich einen Spielzettel hat mit zum Beispiel 6 mal 6 Quadraten aufgedruckt, jedes Quadrat gefüllt mit einem Wort. Ziel des Spieles ist, während eines Gespräches, Vortrages oder Ähnlichem ein vorgekommenes Wort auf seinem Zettel ab zu-haken. Derjenige Spieler welcher zuerst eine waagerechte oder senkrechte Reihe “voll“ hat, gewinnt das Spiel. Die im Spiel verwendeten Wörter passen meist thematisch zum Kontext, in dem das Spiel stattfindet. Zum Beispiel können IT-Fachbegriffe verwendet werden, während eine entsprechende Vorlesung besucht wird.

Als Herausforderung habe ich mir einige besondere Applikationsmerkmale als Ziel gestellt:

- Der Benutzer soll innerhalb der Anwendung bestehende Wortlisten verwalten und erweitern können.
- Jedes Spiel stellt einen Spielraum dar. Spielräume können beliebig oft eröffnet und im nach-hinein betreten werden können.
- Während eines Spiels erhalten die Nutzer Feedback über den Fortschritt ihrer Mitspieler.

Zur Entwicklung habe ich die folgende Technologien verwendet. Namentlich für den Hauptentwurf der Seite HTML5, CSS3 und JavaScript. Es wurde mit Absicht auf jQuery verzichtet um einerseits die Seite etwas schlanker zu halten, aber hauptsächlich als Programmierübung in purem JavaScript. Des weiteren GitHub, als zentrale Ablage für den Code und die Versionierung, Node.js für die Serverkommunikation und MaterializeCSS als Framework, um das Design der Website erheblich zu erleichtern.

Der Programmcode ist auf dem öffentlichen GitHub-Repositorium frei zugänglich.[1]

2 Aufbau der Singlepage

Die Applikation ist in sogenannte Ansichten unterteilt, 4 an der Zahl. Anfangs wurde das Aussehen der Website statisch mit Hilfe des MaterializeCSS Framework vorgearbeitet. Anschließend wurden nach und nach die benötigten Ansichten und natürlich die eigentlichen dynamischen Funktionalitäten implementiert.

[1] `https://github.com/paraRhyme/thisIsABingo`

HTML5 entsprechend besteht die Applikation aus einem *header*, *main* und *footer*-Part. In *main* wird je nach Ansicht der entsprechende Inhalt generiert, *footer* stellt kontextspezifisch die Navigation zur Verfügung und bietet dem Benutzer Feedback, wie viele andere Nutzer und Spielräume auf dem Server vorhanden sind. Diese Anzeige ist dynamisch und aktualisiert sich eigenständig, sobald ein entsprechendes Aktualisierungs-Event stattfindet, zum Beispiel ein Benutzer, der die Website verlässt.

2.1 Startansicht

Die erste Ansicht, welche ein Benutzer zu sehen bekommt und ebenfalls die zentrale Anlaufstelle zur Navigation der Applikation. Von hier aus ist es möglich einen neuen Raum zu erstellen, einen bestehenden Raum beizutreten und die Wortsätze zu konfigurieren. Jeder Benutzer kehrt zu dieser Ansicht wieder zurück, sollte er eine der anderen verlassen.

2.2 Konfigurationsansicht

In dieser Ansicht können die Wortsätze verwaltet werden. Dies beinhaltet das Anzeigen aller existenten Wortsätze, die Auflistung derer beinhalteter Wörter und das Hinzufügen oder Entfernen von Wörtern in einem Satz. Ebenfalls ist es möglich einen neuen Wortsatz zu erstellen.

Die Funktionalität einen Wortsatz auch wieder zu entfernen ist in der Codebase existent, jedoch nicht im Frontend implementiert, da dies den Rahmen des Projektes gesprengt hätte. Zu beachten wären Szenarien wie zum Beispiel das Betreten eines bestehendem Raumes, während der gespielte Wortsatz bereits gelöscht wurde. Aufgrund der dynamischen Generation eines Spieles, lag der Aufwand solche Szenarien zu vermeiden außerhalb meines Projektplans. Gegebenenfalls wäre dies nachführbar, übersteigt jedoch eine simple Eingabedaten-Überprüfung im Frontend.

2.3 Raumansicht

Die Raumansicht ist in Zwei Bestandteile getrennt, namentlich das Erstellen eines neuen Raumes und das Betreten eines bestehenden Raumes.

Beim **Erstellen** sind Drei Merkmale wesentlich:

- Das Auswählen eines zu spielenden Wortsatzes.
- Das Vergeben eines Raumamens, um den Raum identifizierbar zu machen.
- Die Wahl eines Spielernamens für den Benutzer selbst.

Nur wenn diese Drei Merkmale ausgefüllt sind, ist der Vorgang fortsetzbar. Der Benutzer bekommt dynamisch Angaben zur Anzahl der Wörter in einem Wortsatz. Die ein-gegebenen Daten werden an den Server über-geben und der Benutzer gelangt zur

Spielansicht.

Beim **Beitreten** ist keine Vergabe eines Raumnamens möglich, jedoch die Auswahl eines bestehenden Raumes, in welchem der Benutzer spielen möchte. Sollte noch kein Raum erstellt worden sein, so ist keinerlei Eingabe von Daten in dieser Ansicht möglich und nur die Option in das Hauptmenü zurückzukehren verfügbar.

Der Benutzer bekommt Angaben zur Anzahl der bereits beigetretenen Spieler in einem Raum. Ebenfalls werden die ein-gegebenen Daten an den Server über-geben und der Benutzer gelangt zur Spielansicht.

2.4 Spielansicht

Quasi die wichtigste und tatsächlich sinngebende Ansicht für die ganze Anwendung, verantwortlich für die Darstellung des Spielfeldes und die Logik für das Auswählen der Wörter und die Überprüfung ob das Spiel mit dem nächsten Schritt gewonnen ist.

Jeder Spieler im Raum bekommt ein vom Server zufällig angeordnetes Spielfeld. Ein Spielfeld besteht aus insgesamt 36 Wörtern, also sollte im Idealfall ein Wortsatz auch 36 Wörter enthalten.

Enthält ein Satz mehr Wörter, so werden 36 zufällig ausgewählte Wörter aus dem gesamten Satz zu einem Spielfeld generiert. Enthält ein Satz weniger Wörter, so wird der Satz auf mindestens 36 Wörter hoch-multipliziert, zufällig neu angeordnet und zum Schluss auf 36 beschnitten.

Zusätzlich bekommt der Benutzer an der Seite Informationen über seine Mitspieler. Mit jedem getätigten Klick auf ein Wort erhalten alle Teilnehmer im Raum eine Übersicht über sein aktuelles Spielfeld, jedoch ohne seine Wörter anzuzeigen.

Zuletzt wird, sobald ein Spieler gewinnt und ein “Bingo“ erreicht, das Spiel für alle Teilnehmer beendet.

3 Verwendete Technologien

3.1 Unterstützende Software

3.1.1 Atom

Als IDE/Editor für mein Projekt habe ich Atom gewählt. Besonders hilfreich war die leichte Erweiterbarkeit durch Plugins, zum Beispiel das Live-Server Package, welches erheblich beim Testen vom Serverside-Code geholfen hat, ebenso Syntax-Highlighting und Code-Completion.

3.1.2 Firefox Developer Toolbox

Die integrierte Developer Toolbox vom Firefox-Browser war enorm Arbeits-erleichternd beim Entwickeln des Frontends, insbesondere das Inspektions-Werkzeug beim Design von HTML und CSS und die Konsole beim Debugging von Client-seitigem JavaScript.

3.1.3 GitHub

Da ich bei der Entwicklung öfters zwischen Desktop-Computer und Laptop gewechselt habe, war die zentrale Code-Ablage von GitHub wirklich sehr praktisch, ebenso natürlich die Versionskontrolle, auch wenn diese bei so einem kleinen Ein-Mann Projekt nicht gebraucht wurde.

3.2 Frontend Software

3.2.1 MaterializeCSS

> *„A modern responsive front-end framework based on Material Design“*
> —Beschreibung auf der offiziellen Website.[2]

MaterializeCSS ist ein freies Frontend-CSS-Framework, ähnlich dem bekannten und beliebten Bootstrap Framework. Ich habe MaterializeCSS verwendet, um die Entwicklung zu beschleunigen und die GUI visuell ansprechender zu gestalten.

3.3 Backend Software

3.3.1 Node.js

Der Server wurde als Node.js-Application aufgesetzt. Außer den Standard-Packages werden *express* und *socket.io* verwendet. Das *express*-Package stellt die Server-seitige Verfügbarkeit meiner Singlepage HTML sicher und *socket.io* gewährleistet die Kommunikation zwischen Client- und Server-seitigem JavaScript-Code.

4 Abschlusswort

Zuletzt möchte ich noch einige Worte zu dem Projekt verlieren.

Die Umsetzung von Idee in Konzept und schlussendlich Programmcode lief spannend und ohne unerwartete Probleme. Meine selbst gesetzten Ziele konnte ich erfüllen und das Programm um einige Zusatzfunktionen erweitern, welche über die pure Basis-Funktionalität hinausreichen. Ein Beispiel ist das Feedback über Mitspieler während eines laufenden Spieles.

[2] `https://materializeCSS.com`

Ebenfalls konnte ich mit dem Projekt Erfahrung sammeln und kann in Zukunft darauf zurückgreifen, da mi mehrfach bewusst wurde wie einige Code-Umsetzungen effektiver als Andere sind. Zum Beispiel habe ich Anfangs noch beim Erstellen verschachtelter DOM-Elemente viele umständliche Einzelschritte vorgenommen, welche vereinfachbar sind durch andere Methoden wie dem Element-Attribut $innerHTML$.

Auch würde ich gegebenenfalls andere Umsetzungskonzepte in Betracht ziehen, wie zum Beispiel für Node.js eine spezialisierte Template-Engine wie EJS oder das bereits in *express* integrierte Package *pug*. Eine solche Umsetzung würde mir ein eleganteres Design von HTML-Seiten ermöglichen, Redundanz vermeiden und perfekt zu einem Multipage-Konzept mit Routing passen.

Schlussendlich besteht für das Projekt auch noch Potenzial für Erweiterbarkeit. Es gäbe die Möglichkeit noch Ideen wie einen Chat für den Spielraum einzufügen oder das MaterializeCSS-Design für mobile Geräte zu optimieren, welches sehr leicht umsetzbar durch angepasste *div*-Klassen ist.

Ebenfalls besteht Potenzial zur Optimierung, $LESS$ um CSS zu Erweitern, $GULP$ um JavaScript zu komprimieren und optimieren und eine spezialisierte Konfiguration des Node.js-Servers, um eine optimale Produktivleistung zu erhalten.