1. Einleitung

Im Laufe des Forschungsprojekt INTER!ACT wurden bereits im Jahr 2017 Schüler der HTL Krems hinzugezogen um an dem Projekt mitzuwirken. Diese konnten das Projekt weiter voranbringen und es wurde ein erster Prototyp zur Disskusionsplattform INTER!ACT entwickelt. Mittlerweile konnte die Plattform weiterentwickelt werden, jedoch war das nicht das Ende des Projekts. Sozialwissenschaftler kamen auf die Idee, die auf der Diskussionsplattform gesammelten Daten zu analysieren und auszuwerten, dadurch kam es zu einer erneuten Kooperation mit der HTL Krems und zu dieser Diplomarbeit.

1.1. Problemstellung

Die Problemstellung die sich durch die Idee, die Daten zu verwerten ergab war folgende. Die gesammelten Nutzerdaten liegen auf einem Server und sind für keinen Sozialwissenschaftler ohne technischen Hintergrund benützbar. In Folge dieser Problematik, wollten sie die Daten in einer Form, mit der ein jeder Sozialwissenschaftler umgehen kann. Dadurch kam es zu der Aufgabe die Daten per CSV-Download abrufbar zu machen, sowie diese gleich zu visualisieren.

1.2. Team

Das Team dieser Diplomarbeit besteht aus Stefan Humpelstetter und Michael Schrabauer. Neben dem Aufftraggeber sind noch Mag. Stefan Knotzer, MMag. Philipp Homar und ao. Univ.-Prof. Mag. Dr. Elfriede Penz, MAS, Eur. Ph.D. daran beteiligt.

1.3. Auftraggeber

Auftraggeber dieser Arbeit ist ING. DR. Clemens Appl, LL.M., Universitätsprofessor an der Donau-Universität Krems.

1.4. Danksagung

In Folge dieser Diplomarbeit gibt es einige Personen, welchen besonderer Dank gilt. Dem Betreuungslehrer Dipl.-Ing. Dr. Reinhardt Wenzina, welcher die Diplomarbeit vermittelt hat, sowie dem Auftraggeber Univ.-Prof. Ing. Dr. Clemens Appl, LL.M.. Durch seine Kooperation konnten die gewünschten Ziele noch besser umgesetzt und mit dem erwarteten Ergebnis erreicht werden.

1.5. Gender-Hinweis

Um einen besseren Lesefluss zu gewährleisten wird auf die Verwendung beider Geschlechtsformen in der folgenden Arbeit verzichtet. Trotzdem sollte sich dadurch kein Geschlecht diskriminiert fühlen, da immer beide gemeint sind.

1.6. Arbeitsaufteilung

Um eine bessere Übersicht über die Arbeitsaufteilung zu haben zeigt die folgende Tabelle, wer welchen Teil geschrieben hat.

2. Theoretische Grundlagen

2.1. Single-Page Applikationen vs. Multi-Page Applikationen

Die Diplomarbeit *INTER!ACT Data Analysis* ist eine Webapplikation und in allen modernen Browsern abrufbar. Webapplikationen ersetzen immer mehr Desktop-Anwendungen, weil sie oft bequemer in der Benutzung sind und nicht an ein Gerät gebunden. Eine Webanwendung kann man nach belieben auf allen geläufigen Geräten, wie zum Beispiel Mobiltelefon, Tablet und Computer, ausführen und verwenden.

Beim Entwickeln von Webapplikationen gibt es zwei Hauptstrategien: **Multi-Page Applikationen** (MPA) und **Single-Page Applikationen** (SPA). Wie die Bezeichnungen dieser Konzepte schon andeuten geht es um die Anzahl der Seiten (Pages) einer Webanwendung.

In den folgenden Kapiteln werden Single- und Multi-Page Applikation beschrieben und auf deren Vor- und Nachteile eingegangen. Da die Diplomarbeit *INTER!ACT Data Analysis* eine Single-Page Applikation ist und mit dem Typescript-Framework *Angular* programmiert wurde, wird auch im Speziellen auf die Probleme eingegangen die so eine Anwendung mit sich bringt und wie man diesen entgegenwirken kann.

2.1.1. Single-Page Applikation

Eine Single-Page Applikation (SPA) ist eine Applikation die mit einem Browser aufgerufen wird und während der Nutzung nicht neu geladen werden muss. Mit SPAs versucht man vor allem ein möglichst natürliches Browser-Umfeld zu schaffen und die Benutzererfahrung nicht durch andauerndes neues Laden der Seite nach jedem Button-Klick einzuschränken. Fordert der User neue Daten an so werden diese im Hintergrund asynchron nachgeladen. [1]

Die folgende Abbildung zeigt die Grundidee einer Single-Page Applikation. Es gibt eine erste Anfrage an den Server um den gesamten HTML-Code der Seite in den Browser des Users zu laden. Danach werden nur noch im Hintergrund asynchrone Requests an den Server gesendet um Daten nachzuladen.[2]

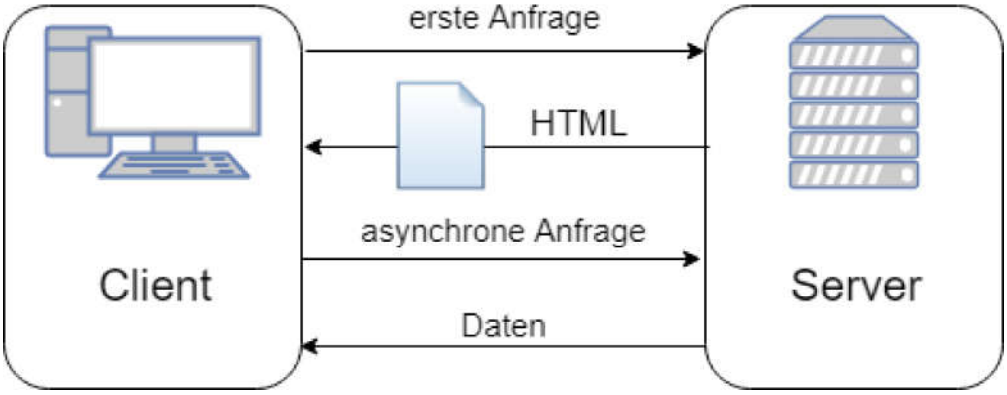

Abbildung 2.1.: SPA Grundidee

2.1.1.1. Vorteile

Ein bedeutender Vorteil der Single-Page Anwendung ist, dass sie sehr schnell ist. Die Applikation wird nur beim Aufrufen einmal geladen und sendet und empfängt danach nur Daten.

SPAs können alle benötigten Daten einmal anfordern und dann auf dem lokalen Speicher zwischenspeichern. Dass ermöglicht es nach einem einzigen Request die Anwendung offline benutzen zu können, was vor allem für User mit schlechter Internetverbindung von Vorteil ist. [3]

Für die Programmierer einer Single-Page Applikation ist es sehr einfach Code für eine native mobile App wieder zu verwenden. Viele SPA-Javascript-Frameworks, wie zum Beispiel *React*, machen Komponenten die für eine Webanwendung erstellt wurden auf anderen Plattformen wiederverwendbar. [4]

2.1.1.2. Nachteile

Es ist schwieriger für Programmierer gute SEO (Search Engine Optimization) zur Verfügung zu stellen. SPAs laden alle Daten asynchron im Hintergrund nach aufrufen der Seite, es ist also sehr schwer eine SPA für Suchmaschinen relevant erscheinen zu lassen. SPA-Frameworks bieten allerdings Möglichkeiten auch eine gute Suchmaschinen-Optimierung umzusetzen. Dass wird anhand von *Angular Universal* im Kapitel 2.1.1.4 genauer erörtert.

Der erste Aufruf einer SPA dauert meist sehr lange, weil die gesamte Anwendung in den Browser des Users geladen wird. Auch für dieses Problem kann *Angular Universal* herangezogen werden.

2.1.1.3. Einsatzszenarien

Die folgenden Szenarien sind typisch für den Einsatz von Single-Page Applikationen.[5]

- **Seiten mit hohen Benutzerzahlen**
  Da bei Single-Page Applikationen ein großer Teil der Rechenleistung clientseitig verrichtet wird bietet sich diese Variante vor allem für Seiten mit vielen Usern an. So erreicht man ein hohes Maß an Skalierbarkeit und eine starke serverseitige Entlastung der Anwendung.
- **Offlineszenarien**
  SPAs können auch offline genutzt werden, wenn alle benötigten Daten bereits im lokalen Speicher geladen sind. Das setzt allerdings vorraus, dass aller weiteren Benutzeraktionen keine neuen Daten benötigen. [6]
- **Kleine Projekte**
  Single-Page Anwendungen eignen sich gut für den Webauftritt von Unternehmen. Für solche Applikationen hält sich die Komplexität der Geschäftslogik meist in Grenzen und ist deshalb als Single-Page Anwendung schnell umsetzbar. Allerdings ist es für Unternehmen sehr wichtig über Suchmaschinen gefunden zu werden, deshalb sollte man für eine single-page Applikation eine Suchmaschinenoptimierung, wie in Kapitel 2.1.4, durchführen.
- **Anwendungen mit hoher Interaktivität**
  Applicationene wie Computerspiele benötigen eine hohe Interaktivität, die bei Multi-Page Applikationen nicht gegeben ist. Früher half man sich mit Plugins wie Adobe Flash oder Microsoft Silverlight aus welche allerdings oft auf mobilen Endgeräten nicht unterstützt wurden. SPAs können sowohl im Browser als auch auf mobilen Geräten aufgerufen werden.

2.1.1.4. Single-Page Applikationen mit Angular

Angular ist ein Typescript-Framework um single-page Applikationen zu erstellen. Angeführt von *Google* wird Angular als Open-Source-Projekt ständig weiterentwickelt und war 2018 das meistgenutzte Frontend-Framework.[7]

Die Hauptbestandteile einer Angular-Applikation sind Komponenten. Angefangen mit einer Root-Komponente besteht eine Angular Anwendung aus vielen, in einander verschachtelten Komponenten. Für Komponenten selbst kann man Darstellung und Verhalten definieren.

Die folgende Abbildung zeigt eine mögliche Struktur von Komponenten in einer Angular-Applikation. [8]

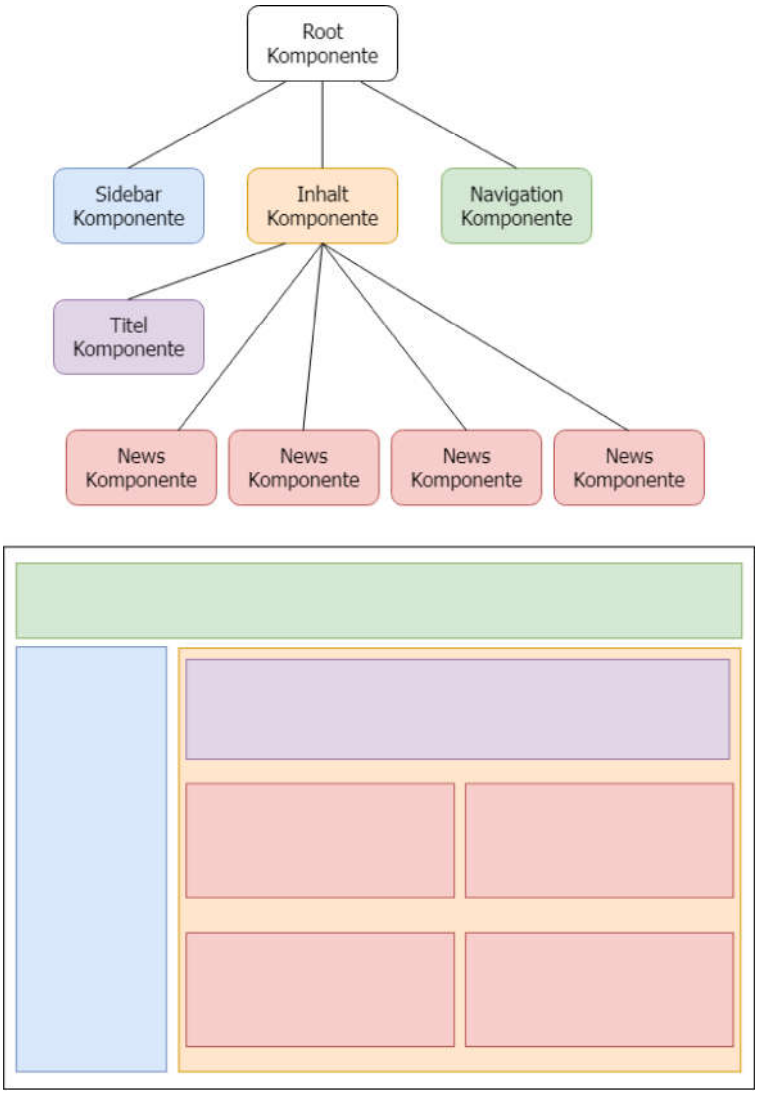

Abbildung 2.2.: Aufbau der Komponenten in Angular

Wie man in der obigen Abbildung sieht können Komponenten beliebig viele Unterkomponenten haben. Angular funktioniert so, dass bei Aufruf der Applikation bereits alle Komponenten in den Browser geladen werden, auch wenn nicht alle gleichzeitig angezeigt werden. Wann welche Komponenten angezeigt werden wird mit TypeScript bestimmt. TypeScript ist eine JavaScript-ähnliche Programmiersprache bestimmt das Verhalten einer Angular-Applikation.

Das Wechseln zwischen den verschiedenen Komponenten passiert in Angular augenblicklich und erfordert kein Neuladen der Seite, was es zu einer single-page Applikation macht.

Eine von Angular zur Verfügung gestellte Komponente ist der *Router*. Diese Komponente zeigt für vordefinierte Routen eine zugeordnete Komponente an. Eine Navigationsleiste setzt man in Angular wie im folgenden Code-Beispiel um. Die *routerLink*-Attribute bestimmen die Route, der *router-outlet*-Tag zeigt je nach Route eine andere Komponente an.

```
<nav>
  <ul>
        <li><a routerLink="/home">Home</a></li>
        <li><a routerLink="/news">News</a></li>
        <li><a routerLink="/kontakt">Kontakt</a></li>
        <li><a routerLink="/impressum">Impressum</a></li>
  </ul>
</nav>

<router-outlet></router-outlet>
 <!-- Content -->
```

Quellcode 2.1: Navigationsleiste als single-page Applikation mit Angular

2.1.2. Multi-Page Applikation

Unter *Multi-Page Applikationen* versteht man traditionelle Webanwendungen die nach jeder Userinteraktion die eine neue Seite laden. Jedes mal wenn ein Benutzer neue Daten anfordert wird eine neue Seite am Server generiert und an den Client gesendet.

Das ständige Anfordern und Senden kann je nach Internetverbindung sehr zeitintensiv sein und führt oft dazu, dass Anwendungen unflexibel sind und dem User keine angenehme Nutzung ermöglichen.

Die folgende Grafik zeigt die Grundidee der Multi-Page Applikation und wie Client und Server zusammenspielen. Grundsätzlich gilt: Für jede Useranfrage wird am Server eine neue HTML-Seite generiert und an den User gesendet.

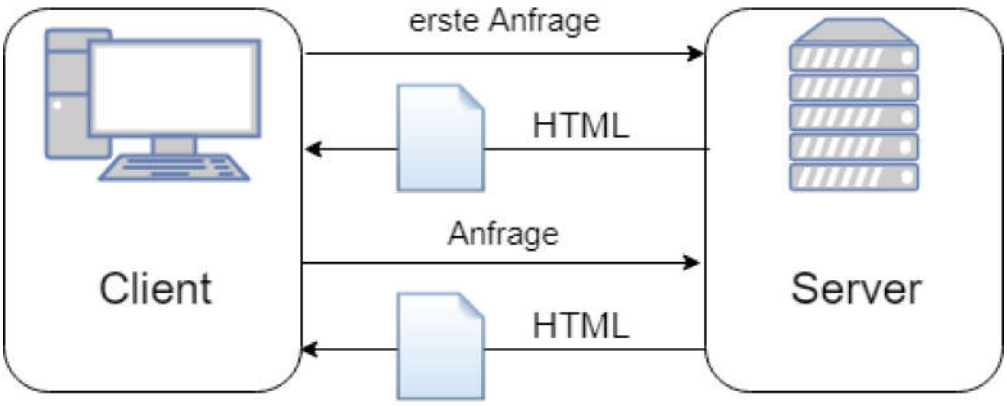

Abbildung 2.3.: MPA Grundidee

2.1.2.1. Vorteile

Einer der Hauptgründe eine Mulit-Page Applikation umzusetzen ist die einfachere Suchmaschinenoptimierung (SEO). Suchmaschinen wie *Google* stufen eine Web-Page nach bestimmten Schlüsselwörtern ein. Dadurch, dass man bei MPAs eine höhere Anzahl an Seiten hat kann man die einzelnen Seiten für ein bestimmtes Schlüsselwort optimieren und hat so eine bessere Chance von Suchmaschinen höher eingestuft zu werden.[9]

MPAs haben außerdem den Vorteil das es sehr leicht ist neue Funktionen und Inhalt in Form von Seiten hinzuzufügen ohne dabei einen großen Performance-Verlust zu riskieren. Deshalb sind Multi-Page Applikationen optimal für Anwendungen mit einer hohen Anzahl an Features geeignet. Sind die Features auf verschiedenen Seiten verteilt werden sie nur dann geladen wenn sie wirklich aufgerufen werden.

Bei Multi-Page Applikationen kann man mit Tools wie *Google Analytics* sehr einfach wertvolle Informationen erhalten, wie zum Beispiel welche Funktionen auf bestimmten Seiten funktionieren und welche nicht. Bei Single-Page Applikationen erhält man mit solchen Tools nicht sehr viele hilfreiche Informationen.

2.1.2.2. Nachteile

Der große Nachteil einer Multi-Page Applikation ist, dass bei jedem Seitenwechsel und bei jeder Datenanforderung die komplette Seite neu geladen werden muss, also auch die Darstellung der Seite (HTML und CSS). [9]

Eine Multi-Page Applikation wird, desto mehr Seiten sie hat, immer weniger Wart- und Erweiterbar. Daraus ergibt sich, dass es schwieriger wird bei der kompletten Anwendung Sicherheit zu gewährleisten, da jede Seite der Applikation einzeln geschützt werden muss.

2.1.2.3. Beispiel

Das folgende Code-Beispiel zeigt anhand einer Navigationsleiste in *HTML (Hypertext Markup Language* wie eine multi-page Applikation funktioniert. Klickt ein Benutzer auf einen der Links wird die jeweils angegebene HTML-Seite heruntergeladen und im Browser angezeigt.

```
1  <!DOCTYPE html>
2  <html lang="de">
3    <head>
4      <meta charset="utf-8"/>
5      <meta name="viewport" content="width=device-width"/>
6      <title>Home</title>
7    </head>
8    <body>
9
10     <nav>
11           <ul>
12                 <li><a href="home.html">Home</a></li>
13                 <li><a href="news.html">News</a></li>
14                 <li><a href="kontakt.html">Kontakt</a></li>
15                 <li><a href="impressum.html">Impressum</a></li>
16           </ul>
17     </nav>
18
19   <!-- Content -->
```

```
20
21     </body>
22 </html>
```

Quellcode 2.2: Navigationsleiste als multi-page Applikation

2.1.3. Vergleich

Um zu entscheiden ob man für eine Webanwendung eine Single- oder Multi-Page Applikation gibt es einige Aspekte die ein Programmierer beachten muss.[10]

- **Internetverbindung**
  Es muss zunächst klar sein wer die Zielgruppe der Anwendung ist und auch wo diese Gruppe lebt. Lebt ein großer Teil der potenziellen Anwender in Gegenden mit schlechtem Internetzugang so ist es besser die Anwendung als Mulit-Page Applikation aufzubauen. Das bringt allerdings nur etwas wenn man nur selten eine neue Seite aufrufen muss, ansonsten wird die Ladezeit in Summe höher sein. In diesem Fall würde man eher eine Single-Page Applikation verwenden. Diese benötigt nur beim ersten Aufruf eine Internetverbindung, sofern danach keine neuen Daten geladen werden müssen.

  Bei der Entscheidung zwischen SPA und MPA muss man also abschätzen wie oft ein User die Seite wechseln wird und die Menge an Daten die während der Benutzung nachgeladen werden müssen.

- **Plattform**
  Entscheidet man sich für den Aufbau seiner Webanwendung als SPA muss im Normalfall ein Backend erstellt werden, das der Single-Page Applikation die Daten zur Verfügung stellt. Möchte man nun die Anwendung auch auf anderen Plattformen zur Verfügung stellen, zum Beispiel als App für mobile Geräte, hat man den Vorteil auf das bereits bestehende Backend zugreifen zu können. Ist die Webanwendung eine Multi-Page Applikation müsste man für eine App zusätzlich ein eigenes Backend erstellen.

  Frameworks mit denen man Single-Page Applikationen erstellt (z.B. Angular und React), bieten außerdem oft die Möglichkeit aus einer bestehenden Webanwendung ohne großem Aufwand eine mobile App zu erzeugen. Bei einer Multi-Page Applikation ist das nicht möglich.

- **Komplexität**
  Ein weiterer wichtiger Aspekt um zu entscheiden ob man eine Single- oder Multi-Page Applikation erstellt ist die Komplexität der Anwendung. Soll die Anwendung eine Vielzahl an zusammenhängender Funktionen bieten ist es besser sie als Single-Page Applikation zu erstellen. SPAs haben im Vergleich zu MPAs eine höhere Wartbarkeit, vor allem bei sehr komplexen Anwendungen, und sind flexibler wenn es darum geht neue Features hinzuzufügen.

  Für simple Anwendungen mit wenigen Features ist es besser eine Multi-Page Applikation zu verwenden, da sie schneller und einfacher zu entwickeln ist.

- **Suchmaschinenoptimierung**
  Suchmaschinen wie *Google* oder *Yahoo* suchen auf Webanwendungen nach bestimmten Schlüsselwörtern und sie für die Suchanfragen ihrer User richtig einstufen zu können. Weil bei Single-Page Applikationen ein großer Teil von relevanten Daten erst nach

dem Aufrufen der Seite nachgeladen wird, ist es für Suchmaschinen schwer sie richtig einzustufen. Bei Multi-Page Applikationen erreicht man leichter eine gute Suchmaschinenoptimierung, weil alle relevanten Daten bereits zum Aufrufzeitpunkt auf der Seite vorhanden sind.
Um auch Single Page Applikationen für Suchmaschinen zu optimieren wird *serverseitiges Rendering (Server Side Rendering)* verwendet. Wie genau das funktioniert wird in Kapitel 2.1.4 erklärt.

Alternativ zu SPA und MPA versucht man bei *hybriden Applikationen* das beste aus beiden Welten zu vereinen und die Nachteile dieser beiden Konzepte möglichst gering zu halten.

Hybrid Applicationen

Mit *hybrid Applications* (hybriden Applikationen) versucht man die Vorteile aus Single-Page und Multi-Page Applikation zu vereinen. Im Normalfall wird eine Multi-Page Applikation verwendet bei der einige Seiten als SPA umgesetzt werden. Das ist vor allem dann eine Option wenn bei der MPA nur auf manchen Seiten komplexere Funktionalitäten benötigt werden. Diese werden dann als Single-Page Applikation umgesetzt. [11]

Um jedoch eine hybride Applikation erfolgreich umsetzen zu können, muss ermittelt werden an welchen Stellen der Applikation die User von einer single-page Anwendung profitieren und bei welchen man es besser bei der MPA belässt. Bei schlechter Umsetzung kann eine hybride Anwendung zu einer mangelhaften Benutzererfahrung und schlechter Performance führen.

2.1.4. Suchmaschinenoptimierung und Server Side Rendering bei Single-Page Applikationen