Anselm Bunsen, Leonard Follner

Technische Dokumentation - Arabismen

Systemvorraussetzungen

Clientseitig

Da es sich um eine Webanwendung handelt, sind die Anforderungen clientseitig den üblichen Anforderungen für WebApps entsprechend. Es wird ein beliebiger moderner Browser benötigt. Auch wenn die Entwicklung für möglichst viele Browser erfolgt ist, kann es vereinzelt zu meist visuellem Fehlverhalten kommen. Daher ist die Verwendung von Chrome sowohl auf Desktops und Laptops alsauch auf mobilen Endgeräten empfohlen, jedoch nicht erforderlich. Die Anwendung ist eine Webanwendung mit Ansätzen, die aus progressiven WebApps bekannt sind. Somit stützt sie sich in ihrer Funktionalität vollständig auf JavaScript. Browser, in denen JavaScript deaktiviert ist, können somit nicht unterstützt werden. Es wird allerdings eine Fehlermeldung angezeigt, die die dieses Problem kurz erläutert.Damit eine Interaktion mit der Anwendung in einer unproblematischen Weise möglich ist, ist eine Mindestdisplaygröße erfoderlich. Die Anwendung prüft selbstständig, ob diese gegeben ist und zeigt eine Warnung an, falls die Größe unterschritten sein sollte. Empfohlen ist ein Display, das mindestens die Größe eines durchschnittlichen Tablets hat.Zum Darstellen der Karte im Hintergrund müssen die Kartendaten von einem Server heruntergeladen werden. Dafür ist eine aktive Internetverbindung zumindest beim ersten Öffnen der Anwendung erforderlich. Für das Laden weiterer Kartenansichten (beispielsweise für die Detailansicht), kann auch während der weiteren Benutzung der Anwendung eine Internetverbindung erforderlich sein. Grundlegend ist die weitere Benutzung auch ohne eine bestehende Internetverbindung möglich. Dann wird die kontextgebende Karte allerdings nicht angezeigt.

Serverseitig

Die einzige serverseitige Voraussetzung ist ein beliebiger installierter Webserver. Dieser muss selbst keine besonderen Bedingungen erfüllen, da keine serverseitigen Berechnungen erfolgen. Er muss lediglich in der Lage sein die entsprechende Menge an Besuchern zu bedienen. Der WebServer kann prinzipiell auch das clientseitige Caching erlauben. Dafür können jegliche Dateien grundlegend beliebig lange clientseitig gecached werden, da der Build-Prozess die Dateien mit content-hashes versieht. Davon ausgenommen werden sollten folgende Dateien, die gar nicht gechached werden dürfen:

_jegliche Bilder im Ordner **`/images`** (es sei denn diese werden nur hinzugefügt und nie geändert)

_die Datei **`data.js`**, da diese die aktualisierbare Datengrundlage bildet (siehe dazu auch Benutzung und Anpassung)

_die Datei **`index.html`**, da über diese die Cache-Invalidierung beim Deployen neuer Versionen erfolgt

_die Datei **`locations.js`** (hierfür gilt das gleiche wie für **`data.js`**)

_die Datei **`main.css`**, da für diese auf Grund des aktuellen Build-Prozesses keine Cache-Invalidierung durch Content-Hashes möglich ist.

Installationsanweisung

vorgefertigter Release

Im ownCloud des Lehrstuhls befindet sich ein vorgefertigter Release der Anwendung. Die Inhalte dies Ordners müssen lediglich in ein Verzeichnis eines WebServers kopiert und online zugänglich gemacht werden.

Release selbst bauen

siehe dazu „Entwicklungshinweise“

Updates

Bei einem Update auf eine neuere Version der Anwendung ist es lediglich erforderlich folgende Dateien zu ersetzen:

_die Datei **build/index.html**

_der Ordner **build/static**

_der Ordner **build/main.css**

Es stellt allerdings auch kein Problem dar, das gesamte Verzeichnis zu ersetzen. Dabei ist allerdings auf eine Sicherung der Inhalte der Dateien **data.js** und **locations.js** zu achten, um Datenverluste zu vermeiden (siehe dazu auch Anpassung der Inhalte).

Bedienungsanleitung

Die Webanwendung erfordert keine speziellen Kenntnisse zur Bedienung. Es muss lediglich mit einem Webbrowser das entsprechende verzeichnis des Webservers aufgerufen werden, in dem die Anwendung installiert wurde. Zu beachten sind dabei die Systemvoraussetzungen. Im unkompilierten Zustand sind auch noch Anpassungen der Funkionalität der Anwendung über gesonderte Steuerflags möglich. Dazu muss vor dem Build-Prozess die Datei app/src/settings.js angepasst werden.

Mögliche Anpassungen sind:

_ export const maxNumberOfCardsOnMap = 3;:

Definiert die Anzahl an Begriffen, die maximal gleichzeitig auf der Karte angezeigt werden.

_ export const maxNumberOfCardsInSideBar = 5;:

Definiert die Anzahl an Begriffen, die maximal gleichzeitig in der Seitenleiste angezeigt werden.

_ export const swipeGestures = false;:

In der Detailansicht kann auf mobilen Endgeräten mittels Wischgesten zwischen den einzelnen Abschnitten gewechselt werden. Diese Funktionalität kann hier (de-) aktiviert werden.

_ export const mobileWarningMayBeIgnored = false;:

Das ändern dieser Einstellung erlaubt es die Warnung, dass das Dispaly des Gerätes zu klein sei, mit einem Kreuz zu ignorieren.

_ export const minimumDeviceWidthNeeded = 750;:

Mittels dieser Eigenschaft kann die Breite eines Displays eingestellt werden, die mindestens benötigt wird, um die Anwendung zu benutzen.

_ export const mapOverlayTransparency = 0.65;:

Dieser Wert beeinflusst prozentual die Transparenz der Flächen, die auf er Karte angezeigt werden.

_ export const mapOverlayColorEurope = ‚#5F9EA0‘; export const mapOverlayColorArab = ‚#DC143C‘;:

_Mittels dieser beiden Werte lassen sich die Farben für die Zielregionen anpassen.

Die hier aufgeführten Werte sind die Standardwerte. Grundlegend sollten alle diese Werte synchron in die Datei **app/src/sass/Utilities/ _ variables.scss** eingetragen werden, da die Änderungen sonst nur teilweise angezeigt werden könnten. Die Syntax entspricht jeweils der von JavaScript und SCSS.

Vorbereitung

Es sind keine gesonderten Vorbereitungen nach der Installation erforderlich. Die Anwendung ist vollständig einsatzbereit. Es ist allerdings empfehlenswert nach der Erstinstallation die Inhalte der Anwendung nach der folgenden Anleitung anzupassen:

Anpassung der Inhalte

Die Inhalte der Anwendung können auch nach dem Build noch angepasst werden. Hierfür sind zwei Dateien entschiedend:

_In **data.js** befinden sich alle Begriffe, deren Definitionen und die Erklärung der Begrisswandlung

_In **locations.js** werden lediglich die einzelnen Orte definiert, die in **data.js** verwendet werden können.

Wichtig ist, dass diese Dateien sowohl im Ordner **app/src/public** und **app/src/build** zu finden sind. Beim Erstellen eigener Builds nach den Installationsanweisungen ist zu beachten, dass die Dateien in **app/src/build** bei jedem Build erneut von denen in **app/src/public** überschrieben werden. Dementsprechend kann es sinnvoll sein, die Dateien außerhalb des Projektes zwischenzuspeichern und erst nach dem Build nach **app/src/build** zu übertragen. Für die Anpassung der Inhalte ist eine grundlegende Kenntnis der JSON-Notation hilfreich. Am Ende der beiden Dateien befinden sich Vorlagen zum Anlegen eines neuen Begriffs oder eines neuen Ortes. Diese können einfach kopiert werden (ohne die Zeilen, die nur **/*** und ***/** beinhalten) und vor der letzten Zeile]; eingefügt werden. Wichtig ist dabei, dass die Klammern erhalten beliben und die Vorlage nicht innerhalb anderer geschweifter Klammern eingefügt wird.

Anpassung der Begriffe

Jeder Begriff entspricht folgender Struktur:

```
{
   „id“: ,
   „name“: „“,
   „arab“: „“,
   „description“: „“,
   „details“: {
     „type“: „sections/map“,
     „changes“: [
     // for sections
       {
         „teaser“: „“,
         „fullDetail“: „“
       },
       // or for map locations
       {
         „location“: „name from locations.js“,
```

```
            „description“: „“
          }
        ]
      },
       „targetRegion“: „Arab“ / „Europe“
    }
```

Die einzelnen Werte müssen dabei wie folgt vor dem Komma eingefügt werden:

_id: eine fortlaufend um 1 inkrementierte Zahl

_name: der Begriff selbst

_arab: die arabische Übersetzung des Begriffs (aus Entwicklungsgründen wird dieser aktuell noch nicht in der Anwendung angezeigt)

_description: eine kurze Beschreibung des Begriffs

_details: beinhaltet innerhalb der geschweiften Klammern alle Schritte der Begiffserklärung in einem von zwei Formaten:

_type: wählt den Typ der Detailansicht, kann entweder „**sections**“ oder „**map**“

_changes:

_wenn als type „sections“ gewählt wurde, muss changes Objekte wie folgt beinhalten:

```
{
    „teaser“: „“,
    „fullDetail“: „“
},
```

_teaser: beschreibt die kurze Überschrift des Textes dieses Abschnittes in der Detailansicht

_fullDetail: beinhaltet den gesamten Text des Abschnittes

_die Abschnitte werden in der Reihenfolge aus dieser Datei angezeigt

_Zur Anpassung der Bilder der einzelnen Abschnitte siehe „Anpassung der Bilder“

_wenn als **type „map“** gewählt wurde, müssen Objekte vom folgenden Typ eingefügt werden:

```
{
        „location“: „name from locations.js“,
        „description“: „“
},
```

_location: ist der Name des Ortes aus der Datei **locations.js**, siehe dazu „Anpassung der Orte“

_description: ist der Text, der neben dem Ort angezeigt wird

_targetRegion: ist die Einordnung des Begriffs in den jeweiligen Sprachraum und kann entweder „Arab“ oder „Europe“ sein.

Anpassung der Orte

Jeder Ort entspricht der folgenden Struktur:

```
{
    „name“: „“,
    „coordinates“: {
      „lat“: ,
      „lng“:
    }
  },
```

Die einzelnen Werte müssen dabei wie folgt vor dem Komma eingefügt werden:

_name: der Name des Ortes, wie er angezeigt werden soll und in der Datei **`data.js`** verwendet wird

_coordinates: beinhaltet die Koordinaten des Ortes auf der Karte

Anpassung der Bilder

Bilder sind im Ordner **`images`** abzulegen. Jeder Begriff erhält dafür einen Unterordner mit dem Namen des Begriffs. Das Bild, das in der Hauptansicht in der Karte angeigt wird und auch als Marker auf der Karte verwendet wird, muss den gleichen Namen wie der Begriff tragen. Falls für die Detailansicht der Modus „sections" in „Anpassung der Begriffe" gewählt wurde, müssen die Bilder den Namen des Begriffs und eine fortlaufende bei 0 beginnende Nummer enthalten. Beide Teile werden durch einen - voneinander getrennt. Die Bilder werden in der Reihenfolge der Nummern für die Abschnitte verwendet. Als Format für die Bilder steht ausschließlich .jpg zur Verfügung.

Benutzung

Grundlegend kann die Anwendung auf jedem Gerät, das den Systemvoraussetzungen entspricht genutzt werden, sobald es Zugriff auf den Webserver hat.

Systemarchitektur

Verzeichnisstruktur

Grundlegend gibt es folgende relevanten Verzeichnisse im Projekt:

`app/build:`

beinhaltet den fertigen Build

`app/public:`

beinhaltet Dateien, die 1-zu-1 in den Build-Ordner kopiert werden

das sind Inhalte und Bilder

`app/src:`

beinhaltet jeglichen Quellcode der Seite

`app/src/Components:`

alle in der Anwendung verwendeten Komponenten

`app/src/Redux:`

das Speichermodell der Anwendung

`app/src/sass:`

die Styles der Anwendung

Hardware

Es wird keine spezielle Hardware verwendet.

Software (Aufbau und Funktionsweise)

Die Anwendung verwendet zum Rendern React.

Das Datenmodell wird dabei von Redux gehändelt.

Die Karte wird von Leaflet gerendert. Das Styling dieser kommt von Mapbox.

Das Styling erfolgt mittels SCSS. Dabei wird eine lose Mischung von BEM und Atomic Design verwendet.

Entwicklungshinweise

Für den Buildprozess werden lediglich Yarn und Sass benötigt. Diese können nach der Anleitung auf der jeweiligen Website oder mittels eines Packagemanagers wie brew installiert werden.

Bauen der Anwendung

Die weitere Installation erfolgt wie folgt:

```
git clone git@gitlab.mg.inf.tu-dresden.de:komplexpraktika/InfoVis/InfoVis_17-18_Arabismen.git
cd app
yarn
```

Der Entwicklungsserver wird ebenfalls mittels yarn gestartet:

```
yarn start
```

Ein Build kann mit folgendem Befehl erstellt werden:

```
yarn build
```

Optional kann die Installation auch mittels **npm** erfolgen. Die entsprechenden Befehle sind **npm run start** für den Developmentserver und **npm run build** für das erstellen eines Builds.

Kompilieren der Styles

Das Kompilieren der Styles ist nicht in den oben genannten Schritten enthalten. Es wird jedoch das Package **node-sass** installiert, dessen Binaries zum Kompilieren der Styles genutzt werden können. Zum Kompilieren der Styles mittels **sass** muss folgender Befehl im Ordner **app** ausgeführt werden:

```
sass src/sass/main.scss build/main.css
```

Für kontinuierliches Neukompilieren geänderter Stylesheets kann folgender Befehl verwendet werden:

```
sass --watch src/sass/main.scss:build/main.css
```

Abschließend kann der erzeugte Ordner app/build zur Installation der Anwendung auf einem WebServer verwendet werden.