Erneuerbare Elektrizitätsproduktion nach Energieträgern und Gemeinden Thurgau 2015-2020

Projektwoche 2i OGD (22.05.2023-26.05.2023)

Verfasser: Simeon Stix

Betreuende Lehrperson: Martin Hager und Silvan Hofstetter

Informatikmittelschule

Abgabetermin: 26.05.2023

Inhaltsverzeichnis

1 Analyse 3
1.1 Aufgabenstellung 3
1.2 Informationenbeschaffung 3
1.3 Ziele 3
1.4 Must-Haves 3
1.5 Nice-to-haves 3
1.6 Zielgruppe 3
1.7 Abgrenzungen 3
1.8 Use Case 3
1.9 Umgebung 4
1.10 Mitarbeiter 4
1.11 Zeiteinteilung 4
1.12 Projekt Antrag Beschreibung 5
2 Design 6
2.1 Mockup 6
2.2 Testfälle 6
2.3 Datenbank Schemata 8
2.4 Rest API Schemata 9
2.4.1 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/[Jahr] 9
2.4.2 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/topAndWorst 10
2.4.3 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/[Jahr]/[GemeindeNR] 10
2.4.4 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/gemeinde/[GemeindeNummer] 10
2.4.5 ./api/polizeiposten 11
3 Coding 11
3.1 Sprache 11
3.2 Ordner Struktur 11
3.2.1 Wichtige Ordner 11
3.3 Frameworks 12
Testen 12
4 Deployment 13
4.1 Installationsanleitung 14

5 Benutzeranleitung 14

1 Analyse

1.1 Aufgabenstellung

Die Aufgabenstellung ist es eine Webapplikation mit einem Backend and Frontend und dabei Statistik Daten des Kanton Thurgau ansprechend darzustellen.

1.2 Informationenbeschaffung

Für die Informationsbeschaffung wird der zugeteilte Datensatz der Statistikdaten Bibliothek verwenden. Um Probleme in der Arbeit zu beheben, wird das Internet zur Hilfe genutzt, konkreter Artikel und Foren jedoch keine künstliche Intelligenz.

1.3 Ziele

Ziel der Applikation ist es die Karte das Kanton Thurgaus, aufgeteilt in die Gemeinden, darzustellen und den entsprechenden Datensatz aus der Open Government Data Bibliothek den Gemeinden zuzuordnen und grafisch darzustellen.

1.4 Must-Haves

- SVG-Karte des Kanton Thurgau als SVG eingefügt
- 3 unterschiedliche Diagramme
- Popup wenn auf eine Gemeinde der Karte geklickt wird
- 2 unterschiedliche Diagrammetypen
- Rest-API
- Datenbank mit ODG-Datensätzen

1.5 Nice-to-haves

- Farbkodierung der Karte
- Idealer Wohnortsuche mit mindestens fünf unterschiedlichen Datensätzen
- -Slider mit dem sich durch die vergangenen Jahre rotieren lässt

1.6 Zielgruppe

Dieses Produkt ist für Statistikinteressiert gedacht die sich genau nur einen Datensatz ansehen wollen. Ansonsten hat dieses Produkt keine weitere Zielgruppe, da dieses Produkt nicht für die Benutzung gedacht ist.

Diese Zielgruppe möchte vor allem sehen, wie der Thurgau sich über die Jahr verändert hat bezüglich des Datensatzes

1.7 Abgrenzungen

Es sind nur ein Datensatz aus der ODG-Bibliothek für unsere Verwendung vorgesehen. Dieser befindet sich in einer JSON-Datei und umspannt die Jahre von 2015 bis 2020.

Nachtrag während der Entwicklung sind auch die Daten für das Jahr 2021 hinzugefügt worden.

1.8 Use Case

USE CASE	BESCHEIBUNG
1	Die Seite wird durch den Nutzer unter der Stamm-URL aufgerufen.
2	Die Seite wird durch den Nutzer unter irgendeiner Sub-URL aufgerufen.

3	Die Karte (Bild des Kantons Thurgau) wird angeklickt.
4	Auf der Sub-Seite „Map“ wählt der Nutzer aus welche Datensätze für ihn interessant ist und kann diesen noch konfigurieren. Bsp.: kann eine maximal/minimal Distanz zum nächsten Polizeiposten einstellen

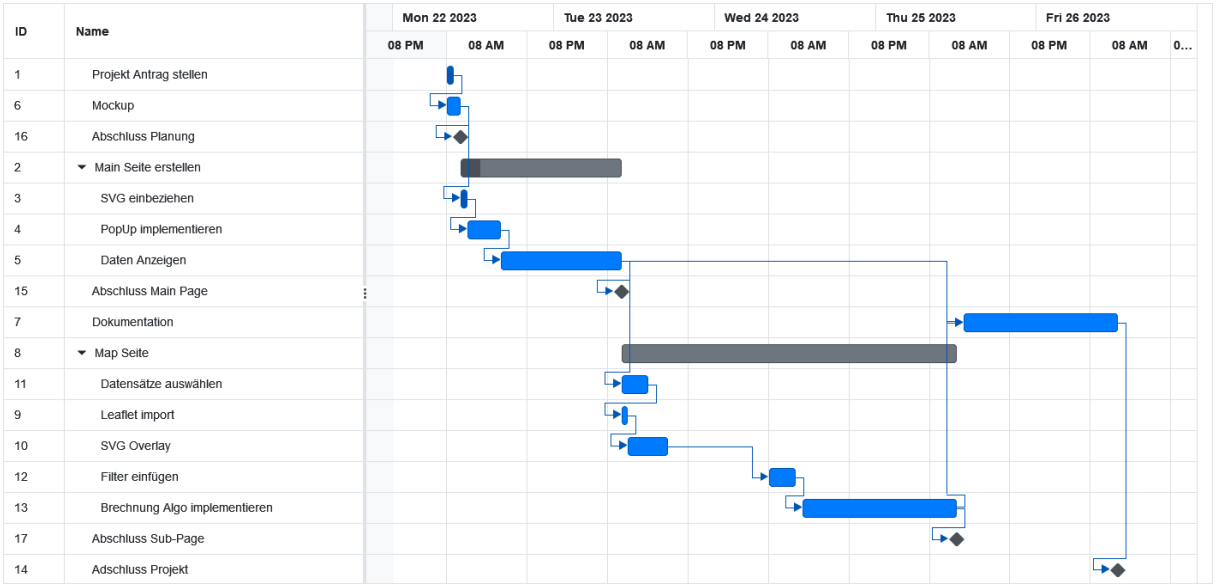
1.9 Umgebung

Die Entwicklung findet in der IDE WebStorm von JetBrains statt. Dabei wird ein NextJS Anwendung entwickelt und somit ein Frontend bestehend aus React Komponenten. Für das Design wird wenig CSS und das Framework Tailwind eingesetzt.

1.10 Mitarbeiter

MITGLIED	ORGAN
SIMEON STIX	Entwicklung

1.11 Zeiteinteilung



AUFGABE	ZEIT
PROJEKT ANTRAG STELLEN	1 Stunde
MOCKUP	2 Stunden
ABSCHLUSS PLANNUNG	Meilenstein
MAIN SEITE ERSTELLEN	8 Stunden = 1 Tag
↪ SVG EINBEZIEHEN	1 Stunde
↪ POPUP IMPLEMENTIEREN	4 Stunden
↪ DATEN ANZEIGEN	3 Stunden
ABSCHLUSS MAIN-PAGE	Meilenstein
DOKUMENTATION	1 Tag
MAP SEITE ERSTELLEN	21 Stunden = 2.6 Tage

↪ DATENSÄTZE AUSWÄHLEN	3 Stunden
↪ LEAFLET IMPORTIEREN	1 Stunde
↪ SVG OVERLAY	5 Stunden
↪ FILTER EINFÜGEN	4 Stunden
↪ ALGORITHMUS FÜR BERECHNUNG VON OPTIMALER POSITION	8 Stunden
ABSCHLUSS MAP-PAGE	Meilenstein
ABSCHLUSS PROJEKT	Meilenstein

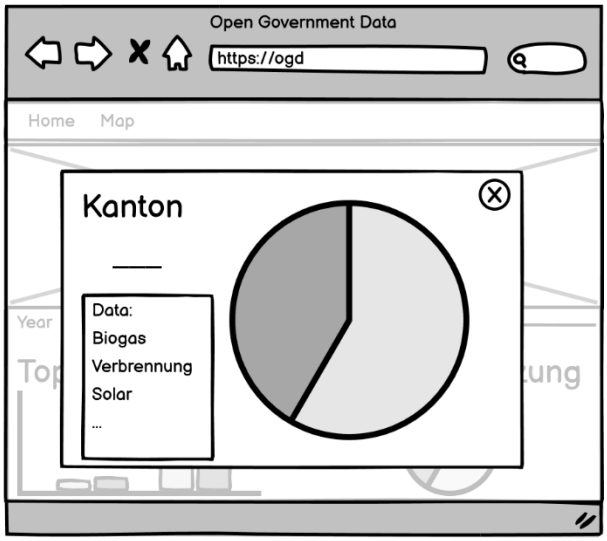
1.12 Projekt Antrag Beschreibung

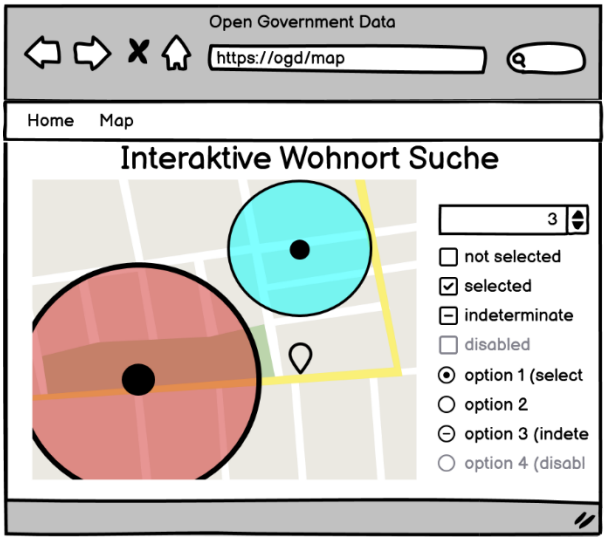
Die Aufgabenstellung ist es eine Webapplikation mit einem Backend and Frontend und dabei Statistik Daten des Kanton Thurgau ansprechend darzustellen.Dies wird erreicht, indem die entsprechenden Datensätze aus der Bibliothek per JSON ausgelesen werden (per fetchAPI angefragt) und danach pro Gemeinde verarbeitet. Dies zeigt sich in der Form, dass hinter dem Bild des Kanton Thurgau eine Funktion verborgen ist die über einen Popup, welches sein Erscheinungsbild bei einem Klick auf die entsprechende Gemeinde zeigt. Dieser Klick wird im Hintergrund verarbeitet. Dies bedeutet das die Daten aus der entsprechenden Datenbank (SQLite) gelesen, an die entsprechenden Stellen des HTML-Strukturcodierung eingefügt werden. Hierbei ließe sich auch noch ein detailliertes Diagramm der erneuerbaren Energiemischung der einzelnen Gemeinden darstellen. Im allgemeinem wird dieses Erscheinungsbild mithilfe von einem „*useState*“ geändert. Eine weiter eventuelle Erweiterung dieses Projekt wäre eine Farbcodierung der Karte, durch welche man die totale prozentuale Nutzung von erneuerbarem Energieträger im absoluten Energiemix darstellt werden und einen Slider mit welcher man die Karte durch die einzelnen Jahre der Datensätze bewegen werden kann.
Im Falle, dass die Applikation vor dem Abschluss der Entwicklungsdauer mit allen Erweiterungen zur Vollendung gebracht wurde und nach dem Ermessen des Entwicklers noch genügend Zeit zur Verfügung steht, wird eine zusätzliche unabhängige Funktionalität hinzugefügt. Hierbei handelt es sich um eine weitere Seite bei der über eine dynamische Karte mehrere Datensätze der Open Government Data Bibliothek dargestellt und zusammengefügt werden. Hierbei ist das Anwendungsziel die Suche der perfekten Gemeinde. Um dieses Ziel zu erreichen kann der Nutzer mithilfe mehreren Filtern Datensätze berücksichtigen und Eingrenzungsoptionen auswählen, welche ebenfalls wenn möglich grafisch in der dynamischen Karte angezeigt werden. Implementierungsmöglichkeiten für eine solche dynamische Karte wird von dem ukrainischen Kartenframework Leaflet zur Verfügung gestellt.

Die gesamte Applikation wird, wie ersichtlich, mit „ReactJS“ aufgebaut. Da jedoch ein Backend gefordert wird, wird das Framework „NextJS“ zum Einsatz kommen. Dementsprechend wird es auch keine dezidierte Rest API entwickelt und/oder zur Verfügung gestellt.

2 Design

2.1 Mockup





2.2 Testfälle

NUMMER	BESCHREIBUNG	TYPE	ERGEBNIS	ERFÜLLT & PROBLEME
1	Der Nutzer ruft die Webseite normal auf.	Konstruktiv	Die Webseite erscheint, das SVG wird mit der richtigen Farbcodierung geladen, es ist auf das letzte Erfassungsjahr eingestellt	
2	Der Nutzer drückt auf die das Feld rechts und unterhalb des Bodensees.	Konstruktiv	Ein Popup erscheint mit allen Inhalten wie im Wireframe festgelegt.	

3	Der Nutzer drückt auf den „Schließen“-Button des Popups	Konstruktiv	Der Popup schließt sich.
4	Der Nutzer macht das Browserfenster kleiner.	Konstruktiv	Die Webseite ändert ihre Größen. Die Diagramme, die bei einem Computer Screen (Fullscreen) nebeneinander liegen, sind nun übereinander.
5	Der Nutzer klickt in der Navigationsleiste auf den Punkt „Map“ (könnte in der Endfassung auch „Karte“ heißen).	Konstruktiv	Die Webseite ändert ihre Seite auf den Kartenansicht.
6	Der Nutzer bearbeitet die Filtereinstellungen.	Konstruktiv	Die Karte mit den Anzeigen ändert sich.
7	Der Nutzer drückt auf den Knopf, „Wohnort berechnen“	Konstruktiv	Ein Pointer wird auf der Karte gesetzt, wo der ideale Wohnort wäre.
8	Der Nutzer ruft die Sub-URL „./test“ auf.	Destruktiv	Der Nutzer bekommt eine personalisierte 404-Fehlerseite angezeigt.
9	Der Nutzer hat es auf die Hauptseite geschafft und klickt auf den Bodensee	Destruktiv	Der Nutzer erhält kein Popup, jedoch erscheint ein freundliches „Alert“, welches den Nutzer darauf aufmerksam macht, dass der Bodensee keine Gemeinde ist.
10	Der Nutzer schafft es zu einem Popup und zu öffnen und möchte es nun schließen. Dazu klickt es nicht auf das Kreuz, sondern direkt neben den Popup.		Auch wenn diese Handlung des Nutzers nicht ideal ist, reagiert das Programm korrekt und schließt den Popup.

11	Der Nutzer schafft es zur „Karten“-Seite und kommt nicht mit den selbsterklärenden Filtern zurecht.	Der Nutzer hat somit Pech. Evtl. wird es in Zukunft eine Hilfs-Seite geben worauf man unter einem Hilfsknopf „ **?** “ weitergeleitet wird.

2.3 Datenbank Schemata

erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden		
nr_gemeinde	String	benötigt
biogasanlagen_abwasser	Float?	nicht benötigt
biogasanlagen_industrie	Float?	nicht benötigt
biogasanlagen_landwirtschaft	Float?	nicht benötigt
biomasse_holz	Float?	nicht benötigt
einwohner	Int?	nicht benötigt
gemeinde_name	String	benötigt
jahr	String	benötigt
kehricht	Float?	nicht benötigt
photovoltaik	Float?	nicht benötigt
wasserkraft	Float?	nicht benötigt
wind	Float?	nicht benötigt
total	Float?	nicht benötigt
polizeiposten		
key	String @unique	einmalig
art	String	benötigt
koordinaten	String	benötigt
schutzraeume		
key	String @unique	einmalig
plaetze	String	benötigt
bau	String	benötigt
koordinaten	String	benötigt
notfalltreffpunkte		
key	String @unique	einmalig
koordinaten	String	benötigt
quellfassungen		
key	String @unique	einmalig
min	String?	nicht benötigt
max	String?	nicht benötigt
koordinaten	String	benötigt

strom		
key	String @unique	einmalig
koordinaten_liste	String	benötigt
risikostelle		
key	String @unique	einmalig
koordinaten	String	benötigt
strassenverkehrzaehlung		
key	Int @id @default(autoincrement())	autoincrement
jahr	String	benötigt
messort	String	benötigt
fahrzeugeProTag	Int	benötigt
koordinaten	String	benötigt
kreisverkehr		
key	String @unique	einmalig
name	String	benötigt
besitzer	String	benötigt
koordinaten	String	benötigt

2.4 Rest API Schemata

Bezüglich der Kodierung:

ZEICHEN	**DEFINITION**
?	Nicht sicher, ob dieses Attribut mitgeliefert wird
[…]	Array (alles, was sich darinnen befindet, befindet sich in Sub-Objekt)
ATTR: [TYPE]	Array vom Typ

2.4.1 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/[Jahr]

```
[
allg
      nr_gemeinde: String
      gemeinde_name: String
      jahr: String
      einwohner: Int
      total: Float
      color: String
traeger
      ?biogasanlagen_abwasser: Float
      ?biogasanlagen_industrie: Float
```

```
      ?biogasanlagen_landwirtschaft: Float
      ?biomasse_holz: Float
      ?kehricht: Float
      ?photovoltaik: Float
      ?wasserkraft: Float
      ?wind: Float
]
```

2.4.2 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/topAndWorst

```
top:
       [
      gemeinde_name: String
      nr_gemeinde: String
      total: Float
      ]
top:
      [
      gemeinde_name: String
      nr_gemeinde: String
      total: Float
      ]
```

2.4.3 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/[Jahr]/[GemeindeNR]

```
gemeinde_name: String
einwohner: Int
?biogasanlagen_abwasser: Float
?biogasanlagen_industrie: Float?
?biogasanlagen_landwirtschaft: Float?
?biomasse_holz: Float
?kehricht: Float
?photovoltaik: Float
?wasserkraft: Float
?wind: Float
total: Float
```

2.4.4 ./api/erneuerbareElektrizitatsproduktionNachEnergietragernUndGemeinden/gemeinde/[GemeindeNummer]

```
gemeinde_name: String
nr_gemeinde: String
data
      [
      jahr: String
      ?biogasanlagen_abwasser: Float
      ?biogasanlagen_industrie: Float
      ?biogasanlagen_landwirtschaft: Float
      ?biomasse_holz: Float
      ?kehricht: Float
      ?photovoltaik: Float
```

```
?wasserkraft: Float
?wind: Float
total: Float
]
```

2.4.5 ./api/polizeiposten

```
key: String
coordinates: Number[]
```

3 Coding

3.1 Sprache

Als Sprache des Projektes wird eine JavaScript und TypeScript Mischung verwendet. Hinzukommen unterschiedliche Konfigurierungsdateien in den Formaten JSON, Prisma, etc. und wenige Assets.

3.2 Ordner Struktur

3.2.1 Wichtige Ordner

ORDNER PFAD	BESCHREIBUNG
ASSETS	Im Asset Ordner befinden sich Dateien welche teilweise in der Applikation als Ressourcen verwendet werden, aber auch Dokumente die nur als Notizen fungieren.
COMPONENTS	Beinhaltet React Komponente, welche auf den einzelnen Webseiten-Seiten verwendet werden.
MODULES	Beinhaltet TypeScript und JavaScript Funtionen und Classen, welche für gewisse Prozess von Nöten sind.
NODE-MODULES	Beinhaltet alle installierten Pakete, welche mit *npm install* installiert werden.
PAGES	Beinhaltet alle Webseiten-Seiten, welche angezeigt werden können und die Grundseite „_app.tsx“
PAGES/API	Beinhaltet alle Rest-API Pfade und Dateien die Aufgerufen werden können.
PRISMA	Beinhaltet die Datenbank und das Schema Skript, mit welchem das Framework Prisma arbeitet.
PUBLIC	Beinhalter alle öffentlich zugänglichen Dateien.
STYLES	Beinhaltet alle CSS Module, welche zusätzlich zu Tailwind angelegt wurden. Tailwind wird über das Dokument „globals.css“ importiert.
TYPES	Beinhaltet alle Type Module, welche man für TypeScript Funktionen anlegen kann.

3.3 Frameworks

Die Applikation ist eine Applikation mit dem NextJS Framework und ist somit im weitersten Sinne eine React Applikation mit integrierten Backend.
Für die Datenbank Verknüpfung wird das Framwork Prisma verwendet, welches support für die unterschiedlichsten Datenbanken bieten würde. In diesem Fall wird eine SQLite Datenbank verwendet.
Bezüglich CSS wird das Framework Tailwind verwendet. Dieses wurde erst später hinzugefügt (Es hat Bootstrap ersetzt.) und umfasst nicht alle Design Aspekte.

Testen

NUMMER	BESCHREIBUNG	TYPE	ERGEBNIS	ERFÜLLT & PROBLEME
1	Der Nutzer ruft die Webseite normal auf.	Konstruktiv	Die Webseite erscheint, das SVG wird mit der richtigen Farbcodierung geladen, es ist auf das letzte Erfassungsjahr eingestellt	Erfüllt
2	Der Nutzer drückt auf die das Feld rechts und unterhalb des Bodensees.	Konstruktiv	Ein Popup erscheint mit allen Inhalten wie im Wireframe festgelegt.	Erfüllt
3	Der Nutzer drückt auf den „Schließen“-Button des Popups	Konstruktiv	Der Popup schließt sich.	Erfüllt
4	Der Nutzer macht das Browserfenster kleiner.	Konstruktiv	Die Webseite ändert ihre Größen. Die Diagramme, die bei einem Computer Screen (Fullscreen) nebeneinander liegen, sind nun übereinander.	Erfüllt
5	Der Nutzer klickt in der Navigationsleiste auf den Punkt „Map“ (könnte in der Endfassung auch „Karte“ heißen).	Konstruktiv	Die Webseite ändert ihre Seite auf den Kartenansicht.	Nicht erfüllt
6	Der Nutzer bearbeitet die Filtereinstellungen. (Seite /map)	Konstruktiv	Die Karte mit den Anzeigen ändert sich.	Erfüllt
7	Der Nutzer drückt auf den Knopf,	Konstruktiv	Ein Pointer wird auf der Karte gesetzt, wo	Erfüllt

	„Wohnort berechnen“ (Seite /map)		der ideale Wohnort wäre.	
8	Der Nutzer ruft die Sub-URL „./test“ auf.	Destruktiv	Der Nutzer bekommt eine personalisierte 404-Fehlerseite angezeigt.	Erfüllt
9	Der Nutzer hat es auf die Hauptseite geschafft und klickt auf den Bodensee	Destruktiv	Der Nutzer erhält kein Popup, jedoch erscheint ein freundliches „Alert“, welches den Nutzer darauf aufmerksam macht, dass der Bodensee keine Gemeinde ist.	Erfüllt
10	Der Nutzer schafft es zu einem Popup und zu öffnen und möchte es nun schließen. Dazu klickt es nicht auf das Kreuz, sondern direkt neben den Popup.	Destruktiv	Auch wenn diese Handlung des Nutzers nicht ideal ist, reagiert das Programm korrekt und schließt den Popup.	Erfüllt
11	Der Nutzer schafft es zur „Karten“-Seite und kommt nicht mit den selbsterklärenden Filtern zurecht.	Destuktiv	Der Nutzer hat somit Pech. Evtl. wird es in Zukunft eine Hilfs-Seite geben worauf man unter einem Hilfsknopf „ **?** “ weitergeleitet wird.	Nicht erfüllt

4 Deployment

Es gibt keine Rest API, da die „Index“ Seiten bei dieser Anwendung mit NextJS mit Static Site Generation gerendert werden.

Es könnte noch eine Rest API hinzukommen aber nur auf einer „Nice-To-Have“-Basis für die „Karten“-Seite, da diese wahrscheinlich Client-Side Rendering nutzt und nicht Static Site Generation oder Pre-Rendering.

Als Ergänzung sei erwähnt, dass mir beim Verfassen dieses Abschnittes nicht in Gedanken war, dass, um die lokalen gesicherten Zwischenspeicherdateien möglichst schmal zu halten, nicht den Datensatz in seiner kolossalischen Gesamtheit der angehäuften Aufzeichnungsjahre, sondern nur das gesuchte Jahr und bei potenziellen Optimierungen nur die allfällige Gemeinde einzulesen und anzeigen zu lassen, eine Static Site Generation nicht möglich oder zu mindestens nicht praktikable sei, da diese bei dies beim laden des Datensatz, um mit dem geplanten Slider annoncier bar auf der Karte zu sein, gänzlich und in summa im Stöberer zwischengespeichert werden müsse, welches den Nebeneffekt hätte,

dass die gewonnene Geschwindigkeit, durch die sich nicht wiederholende Berechnung des Darstellungsbereiches infolge der Größe des einmalig gestalteten interaktiven Grafikendeskriptionsterritorium, aufgrund der Übertragungsrate des arrangierten Anzeigedokumentes, verloren gehen würde. In Angesicht dieser Tatsache kommt jedoch auch kein Server Side Rendering in Frage, da der Datensatz zum einen keine fluktuierenden Eigenschaften aufweist, der Vorteil der Zersplitterung des Datensatzes in subpartikulare Datensplitter durch regelmäßiges Verrichten des Gestaltungsprozesses nicht erreicht werden kann und dies ebenfalls für die gemischte Variant, Incremental Static Regeneration, zwischen Server Side Rendering und Static Site Generation gilt, bleibt nur die die Option des Client Side Rendering oder Client Side Fetching. Hierfür wiederum wird eine Rest API benötigt, infolgedessen, dass eine alleinig Static Site Generation zwar möglich aber fachlich unelegant wäre, veranlasse ich eine Revision meiner geistigen Einschätzung und Neubewertung des gewählten Ausgabemodels infolge einer gezielten Änderung der Parameter.

Dementsprechend sind die Entwürfe für die Rückgabewerte der Rest API im Still von klassischen Schemata zu finden.

4.1 Installationsanleitung

Nach dem Klonen der Version Control muss *npm install* durchgeführt werden, damit die benötigten Pakete installiert werden.

An sich ist im package.json bereits unter scripts/build die zwei benötigten Befehle aufgeführt. Somit wird der Bauprozess vereinfacht, *npm run build* ausführen muss, um die Applikation danach auszuführen muss der Befehl *npm run production* ausgeführt werden.

Sollte man keine Produktionsumgebung wollen, lässt sich mit dem Befehl *npm run dev* oder *npm run start* die Applikation in einer Entwicklerumgebung starten.

Bezüglich Migration der Datenbank muss man sich keine Gedanken machen, da im Befehl *npm run build* die Migration automatisch durchgeführt und zusätzlich ist im Server eine Funktion vorhanden, die beim Start die Datenbank überprüft wird und wenn sie diese Prüfung nicht besteht, wird automatisch eine Migration durchgeführt.

5 Benutzeranleitung

Es gibt zwei Hauptelemente auf der Webseite. Zum einem das Stammverzeichnis, in dem sich der Hauptteil der Applikation befindet und einem zweiten Teil der sich unter „/map“ befindet. Dieser zweite Teil ist ein „Proof of Concept“ und somit wurden aus geplanten 10 Datensätzen nur einer und das UI ist nicht fertig ausgearbeitet. Jedoch wäre die Seite abgesehen von den Filtern leicht auf mehrere Datensätze hochzuskalieren.
Zusätzlich gibt es pro Jahr nach einer Seite welche unter „/*jahr*“ aufgerufen werden kann. Diese beinhaltet nur die Thurgau Karte mit der Farbkodierung und werden beim Serverstart statisch gebildet und sehr minimalistisch gehalten werden. Sie sind nicht für die Verwendung gedacht.