1. Installation

download the repository

```
git clone git@github.com:SebastianKapunkt/german-election-viewer.git
```

cd into the project

```
cd german-election-viewer
```

install requirements

```
german-election-viewer$: pip install -r requirements.txt
```

install node dependencies

```
german-election-viewer$: cd WebApp
german-election-viewer/WebApp$: npm install
```

build angular2 typescripts

```
german-election-viewer/WebApp$: npm start
```

set envirement varaible

```
german-election-viewer$: export FLASK_APP=app/__init__.py
```

start the flask server from root folder

```
german-election-viewer$: flask run
```

open localhost

```
http://localhost:5000/
```

2. Parsing btw17_kerg.csv

Before we started to parse we had to figure out how the csv is structured. To make it more human readable we opened it in 'Numbers'.

Identify rows

First we identified what a federal state is and what a constituency.

Nr	Gebiet	gehört zu	Wahlberechtigte		
294	Ravensburg	8	186901	185006	
295	Zollernalb – Sigmaringen	8	183202	184370	
8	Baden-Württemberg	99	7732570	7689895	
296	Saarbrücken	10	199887	204905	
297	Saarlouis	10	207500	211529	
298	St. Wendel	10	177468	181521	
299	Homburg	10	192408	198117	
10	Saarland	99	777263	796072	
99	Bundesgebiet		61675529	61946900	

1. A **federal state** has at column "gehört zu" the number '99'.
2. A **constituency** has at column "gehört zu" a number between 0-17
3. Also with looking further we found that the **state** (eg. germany) is to identify when the column "Nr" had the value '99' and the column "gehört zu" was empty.

Identify columns

The next step is to figure out the structure of the columns after "gehört zu"

t zu	Wahlberechtigte				Wähler				Ungültige	
	Erststimmen		Zweitstimmen		Erststimmen		Zweitstimmen		Erststimmen	
	Vorläufig	Vorperiode	Vorläufig	Vorperiode	Vorläufig	Vorperiode	Vorläufig	Vorperiode	Vorläufig	\
1	225659	226944	225659	226944	171905	162749	171905	162749	1647	
1	186384	186177	186384	186177	139200	131527	139200	131527	1299	
1	175950	176731	175950	176731	132016	126409	132016	126409	1133	
1	199632	198903	199632	198903	157387	149583	157387	149583	1285	
1	204650	205243	204650	205243	151463	146452	151463	146452	1657	
1					131710	127093	131710	127093	1221	
1		0			187711	179055	187711	179055	1616	
1	245826	244240	245826	244240	193318	183250	193318	183250	1628	

1. The type or name of block. This is usally the name of the party where the numbers belong to. Except for the first 4 cases where it is a more general aggregation: Wahlberechtigte, Wähler, Ungültige & Gültige.
2. Each block includes 4 numbers.
    1. "Erststimmen Vorläufig"
    2. "Erststimmen Vorperiode"
    3. "Zweitstimmen Vorläufig"
    4. "Zweitstimmen Vorperiode"

parsing

1. The first thing we do is getting the name of all "Party"-blocks. We just paste the 4th row into an array.

```
def get_parties(rows):
    parties = []
    index = 3
    while index < len(rows[2]) - 1:
        parties.append(Party(name=rows[2][index]))
        index += 4
    return parties
```

2. Now we can go row by row and apply what we figured out to identify which row is what.
    1. federal state

    ```
    elif int(row[2]) == 99:  # federal_state it self
    ```

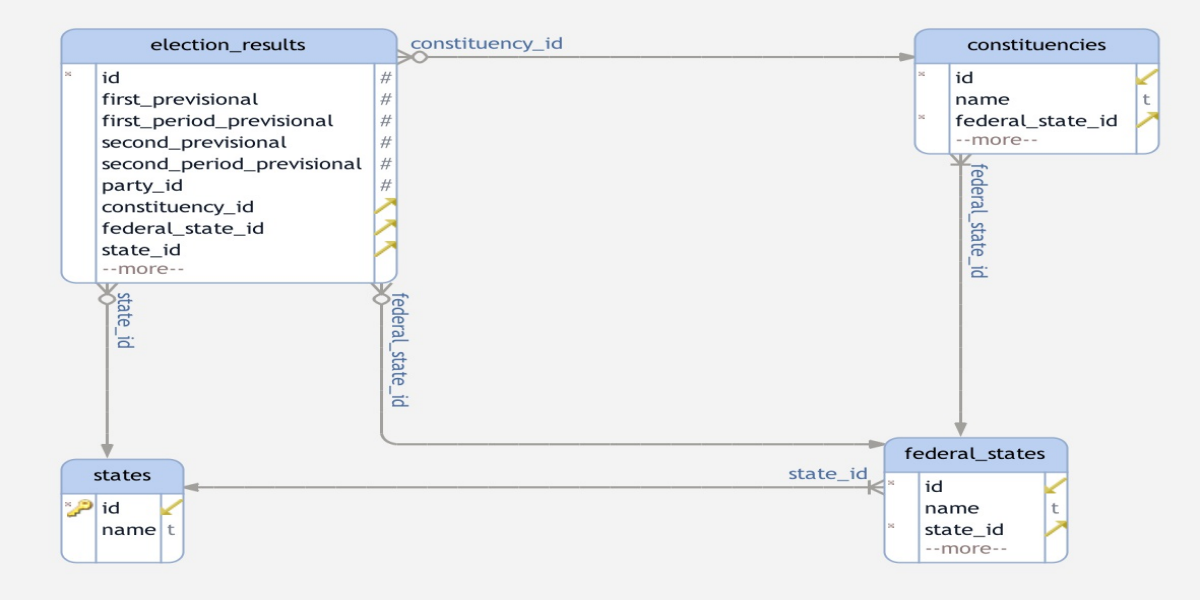
    2. constituency

    ```
    elif int(row[2]) in range(17): #constituencies
    ```

    3. the state

    ```
    if row[2] == '' and row[0] == '99':  # state it self
    ```

3. We have the names of the parties and can identify each row. To continue we had to setup a database model.



4. How all this works together can be seen in election_reader.py

3. Representation as View

Once all the Information were getherd it was important to identify which of those were needed and how they should be formated for the frontend. For this application the frontend which will be a single page application, will be implemented with the latest Angular 2 version.

API Calls

To create a single page application several API calls had to be implemented which will supply the data on demand to the application.

/api/states
Will return all states of Germany

/api/states/<int:federal_state_id>/constituencies
Will return the constituencies of a chosen state

/api/constituency/<int:constituency_id>/general_election_results
Will return the general information regarding the election of a chosen constituency.

/api/constituency/<int:constituency_id>/party_election_results
Will return the election results for the parties of a chosen constituency.

Handling and Parsing the Data

The data which will be queried from a SQL Database need to be parsed into json objects so it can be used in the frontend.

```
states_data = []
```

```
        all_states = Federal_State.query.all()

        for state in all_states:
            states_data.append({'id': state.id, 'name': state.name})

        return jsonify({'data': states_data})
```

The code snipped above will take the information queried, create an array of objects with the attributes id and name and send a json response like:

```
[
    data: {
        {
            id: 1,
            name: 'Schleswig-Hollstein'
        },
        ...
    }
]
```

to the browser.

Using the Data

To use the information create above, we implemented two service classes for the front end.

1. state.service.ts
2. region.service.ts

The state.service will do all the queries regarding states.

The region.service on the other hand will query all the information focusing on regions ('constituencies').

To elaborate on the states example, the front end http get request for the api url: /api/states looks as follows:

```
getStates(): Promise<State[]> {
    return this.http.get('/api/states', { headers: this.jwt() })
        .toPromise()
        .then(response => response.json().data)
        .catch((error) => console.log('no states: ', error));
}
```

This function will emit a http.get request to the specified url and provide the created data from the backend as a json object, if anything goes wrong during this process the function will output an error.

Displaying the Data

The whole app will consist of smaller components which will use the data supplied by the services and represent each segment of the webpage.