Database systems: Assignment Project

Lecturers: Prof. Dr. A. Voisard, M.-U. Karagülle
Tutor: Alexander Stage
Students : Ivan Timofeev, Konstantinos Kokkalis, Meri Sukiasyan, Dea Karam
Github repo with data: https://github.com/oduwancheekee/DBS_project

Task 1: Project Idea

We had a look at the proposed datasets (Fahrraddiebstahl.csv, lor_planungsraeume.csv, and bezirksgrenzen.csv), which could be organized in “bike-theft in Berlin” database. CSV tables contain information on specifically when, where a bike was stolen and some information about a bike (estimated cost, type, comment). With given tables we decided that most informative visual output would be connected to counts of stolen bikes and damages depending on the time and place of a theft. Specifically, we’ll look into:

1. total damage per borough, which could be nicely plotted into pie chart
2. quantity of stolen bikes per area per borough, to be presented in bar chart
3. quantity of bikes stolen per day in Berlin, which is the best visualizable with line plot, as timeline spans for one and a half year
4. quantity of bikes stolen in each planning area, plotted to the map. In order to do this, we’ll enrich our lor_planungsraeume table with geo coordinates of according planning areas.

Task 2: Data schema and database set up

In order to understand the relations of the table we created a meaningful data schema for the data sets and drew an according Entity-Relationship model (ERM) (Figure 1).

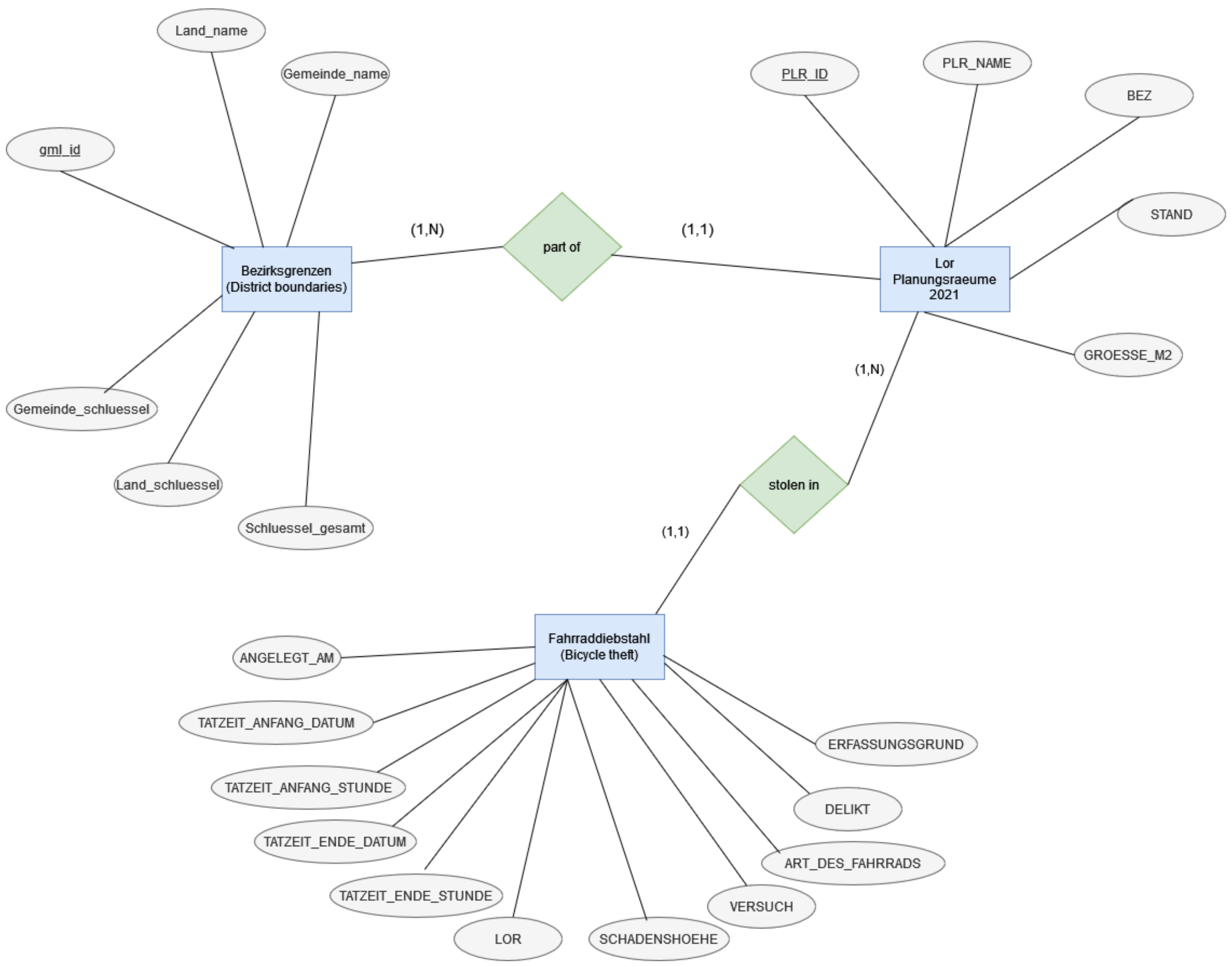


Figure 1. Entity-Relationship model (ERM) for the given datasets in min-max notation.

ERM could be then transformed into the following relational model:

Bezirksgrenzen (gml_id, Land_name, Gemeinde_name, Gemeinde_schluessel, Land_schluessel, Schluessel_gesamt)
Lor_Planungsraeume (PLR_ID, PLR_NAME, BEZ, STAND, GROESSE_M2)
Fahrraddiebstahl (ANGELEGT_AM, TATZEIT_ANFANG_DATUM, TATZEIT_ANFANG_STUNDE, TATZEIT_ENDE_DATUM, TATZEIT_ENDE_STUNDE, LOR, SCHADENSHOEHE, VERSUCH, ART_DES_FAHRRADS, DELIKT, ERFASSUNGSGRUND)
Part_of (gml_id, PLR_ID)
Stolen_in (LOR)

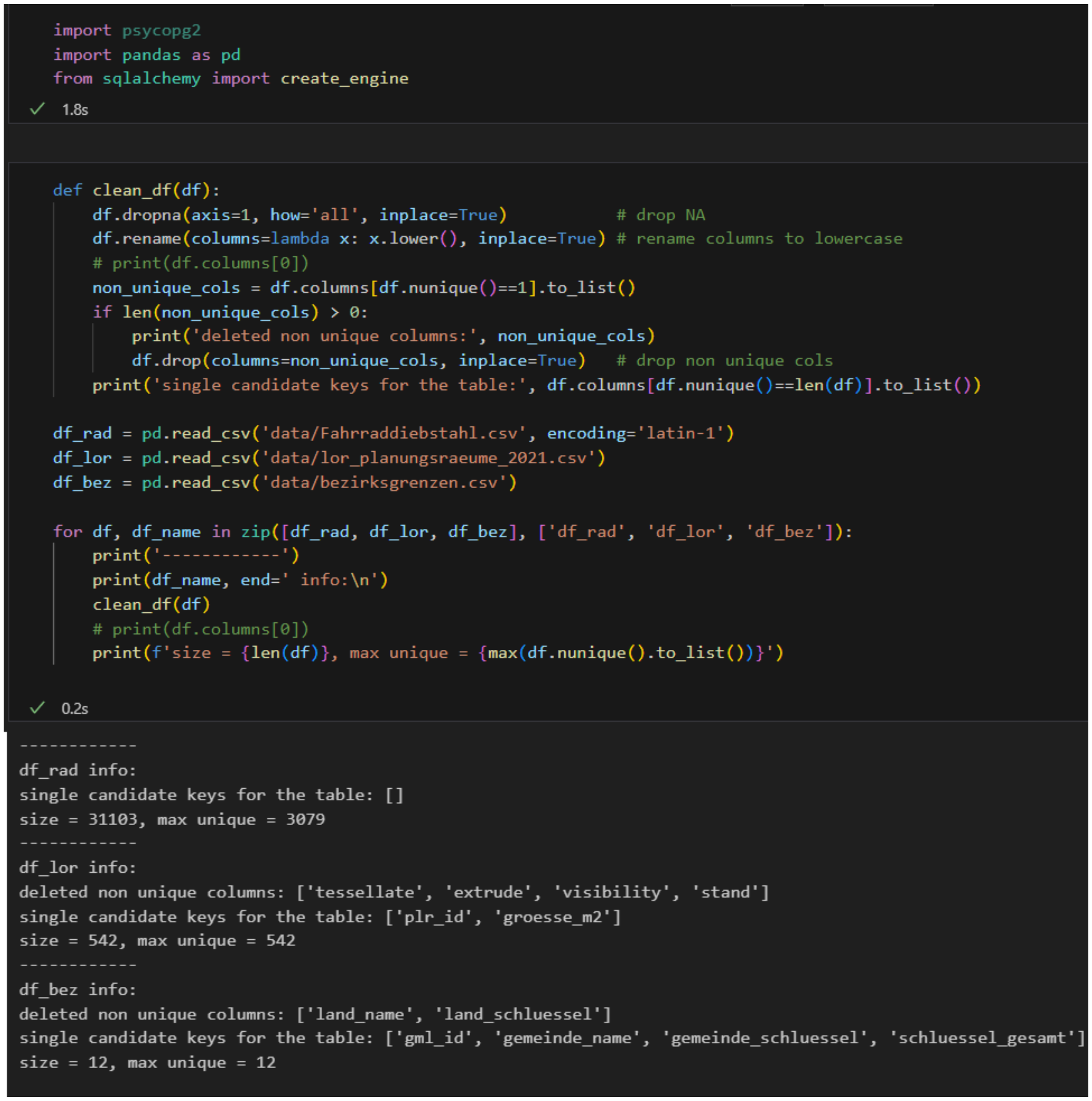
Tables Bezirksgrenzen and Lor_Planungsraeume have several candidate keys, the chosen ones are underlined in the model. Table Fahrraddiebstahl doesn’t have any candidate key.

To set up a database we employed Postgresql DBS. Here are the Ubuntu specific instructions, based on our approach:

- install postgresql: sudo apt-get install postgresql
- ensure its running: sudo service postgresql restart
- create db: createdb fahrraddiebstahl

Task 3: Pre-processing and import data

After creating and pondering on ERM, relational model and setting up a database we employed python and jupyter notebook for cleaning and importing data. In the following are the snippets or preprocessing, gathering some statistics and importing the data:



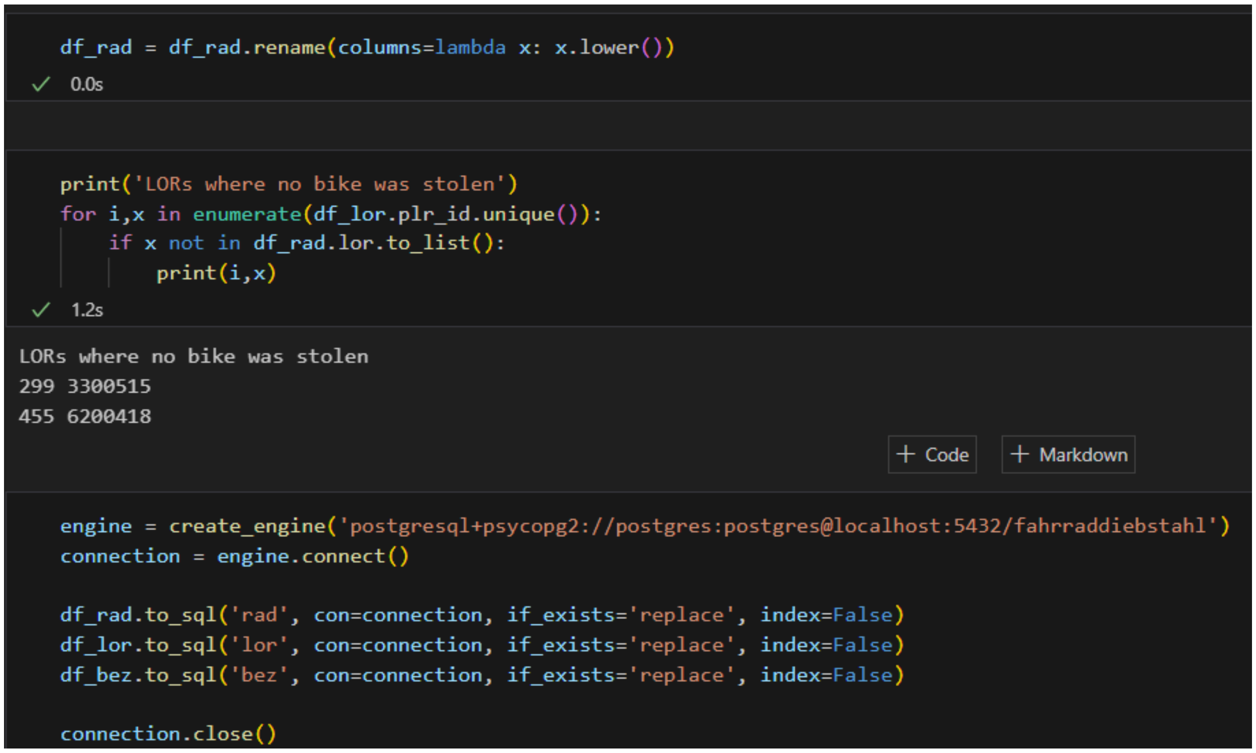
```python
import psycopg2
import pandas as pd
from sqlalchemy import create_engine
```

✓ 1.8s

```python
def clean_df(df):
    df.dropna(axis=1, how='all', inplace=True)           # drop NA
    df.rename(columns=lambda x: x.lower(), inplace=True) # rename columns to lowercase
    # print(df.columns[0])
    non_unique_cols = df.columns[df.nunique()==1].to_list()
    if len(non_unique_cols) > 0:
        print('deleted non unique columns:', non_unique_cols)
        df.drop(columns=non_unique_cols, inplace=True)   # drop non unique cols
    print('single candidate keys for the table:', df.columns[df.nunique()==len(df)].to_list())

df_rad = pd.read_csv('data/Fahrraddiebstahl.csv', encoding='latin-1')
df_lor = pd.read_csv('data/lor_planungsraeume_2021.csv')
df_bez = pd.read_csv('data/bezirksgrenzen.csv')

for df, df_name in zip([df_rad, df_lor, df_bez], ['df_rad', 'df_lor', 'df_bez']):
    print('------------')
    print(df_name, end=' info:\n')
    clean_df(df)
    # print(df.columns[0])
    print(f'size = {len(df)}, max unique = {max(df.nunique().to_list())}')
```

✓ 0.2s

```
------------
df_rad info:
single candidate keys for the table: []
size = 31103, max unique = 3079
------------
df_lor info:
deleted non unique columns: ['tessellate', 'extrude', 'visibility', 'stand']
single candidate keys for the table: ['plr_id', 'groesse_m2']
size = 542, max unique = 542
------------
df_bez info:
deleted non unique columns: ['land_name', 'land_schluessel']
single candidate keys for the table: ['gml_id', 'gemeinde_name', 'gemeinde_schluessel', 'schluessel_gesamt']
size = 12, max unique = 12
```



```
df_rad = df_rad.rename(columns=lambda x: x.lower())
```
✓ 0.0s

```
print('LORs where no bike was stolen')
for i,x in enumerate(df_lor.plr_id.unique()):
    if x not in df_rad.lor.to_list():
        print(i,x)
```
✓ 1.2s

```
LORs where no bike was stolen
299 3300515
455 6200418
```

```
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/fahrraddiebstahl')
connection = engine.connect()

df_rad.to_sql('rad', con=connection, if_exists='replace', index=False)
df_lor.to_sql('lor', con=connection, if_exists='replace', index=False)
df_bez.to_sql('bez', con=connection, if_exists='replace', index=False)

connection.close()
```

Figure 2. Ipynb snippets or preprocessing, gathering statistics and importing the data. The corresponding code could be found by the link: https://github.com/oduwancheekee/DBS_project/blob/main/db_processing.ipynb

The obtained database could be reached for SQL querying by the command:

```
> psql -U postgres -d fahrraddiebstahl
```

The obtained database could be dumped to file by the command:

```
> pg_dump -U postgres -d fahrraddiebstahl > fahrraddiebstahl_dump.sql
```

The obtained database could be restored locally using dump file by the command:

```
> psql -U postgres -d fahrraddiebstahl < fahrraddiebstahl_dump.sql
```

Task 4: Development of a Web application

The graphs and according SQL queries employed to fetch necessary data from DB are:

1. pie chart: total damage per borough

```
SELECT bez.gemeinde_name AS bezirk,
        SUM(rad.schadenshoehe) AS damage_per_borough
```

```sql
FROM bez
JOIN lor ON bez.gemeinde_schluessel = lor.bez
JOIN rad ON lor.plr_id = rad.lor
GROUP BY bez.gemeinde_name
ORDER BY damage_per_borough DESC;
```

2. bar chart: quantity of stolen bikes per area per borough

```sql
SELECT bez.gemeinde_name AS bezirk,
            COUNT(*)/SUM(lor.groesse_m2)*1000000 AS quantity_per_area_per_borough
FROM bez
JOIN lor ON bez.gemeinde_schluessel = lor.bez
JOIN rad ON lor.plr_id = rad.lor
GROUP BY bez.gemeinde_name
ORDER BY quantity_per_area_per_borough DESC;
```

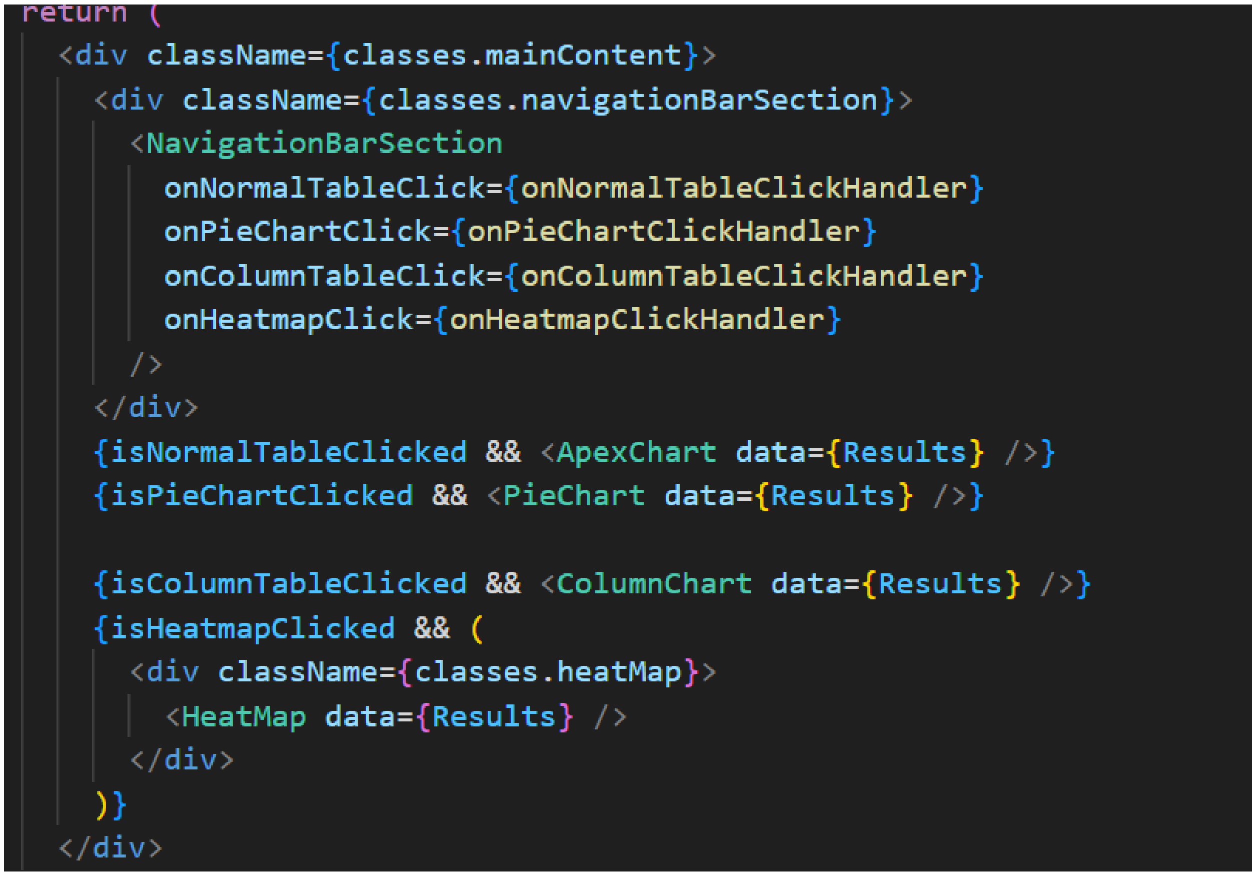
3. line plot: bikes stolen per day

```sql
SELECT TO_DATE(angelegt_am,'DD.MM.YYYY') AS day,
             COUNT(*) AS bikes_per_day
FROM rad
GROUP BY day
ORDER BY day;
```

4. map plot

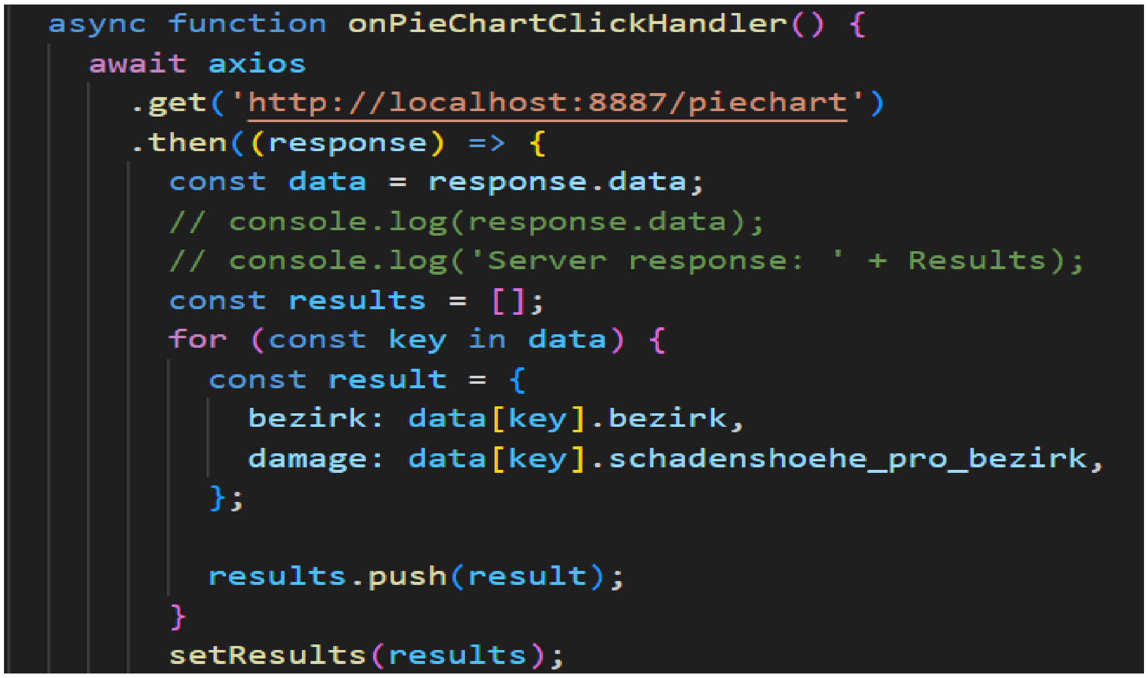
```sql
SELECT lor.plr_name AS planning_area, count(*)
FROM lor
JOIN rad ON lor.plr_id = rad.lor
GROUP BY lor.plr_name
ORDER BY count(*) DESC
LIMIT 50;
```

FRONTEND:
Technologies: React JS
Step 1: Create a User Interface



```
return (
  <div className={classes.mainContent}>
    <div className={classes.navigationBarSection}>
      <NavigationBarSection
        onNormalTableClick={onNormalTableClickHandler}
        onPieChartClick={onPieChartClickHandler}
        onColumnTableClick={onColumnTableClickHandler}
        onHeatmapClick={onHeatmapClickHandler}
      />
    </div>
    {isNormalTableClicked && <ApexChart data={Results} />}
    {isPieChartClicked && <PieChart data={Results} />}

    {isColumnTableClicked && <ColumnChart data={Results} />}
    {isHeatmapClicked && (
      <div className={classes.heatMap}>
        <HeatMap data={Results} />
      </div>
    )}
  </div>
```

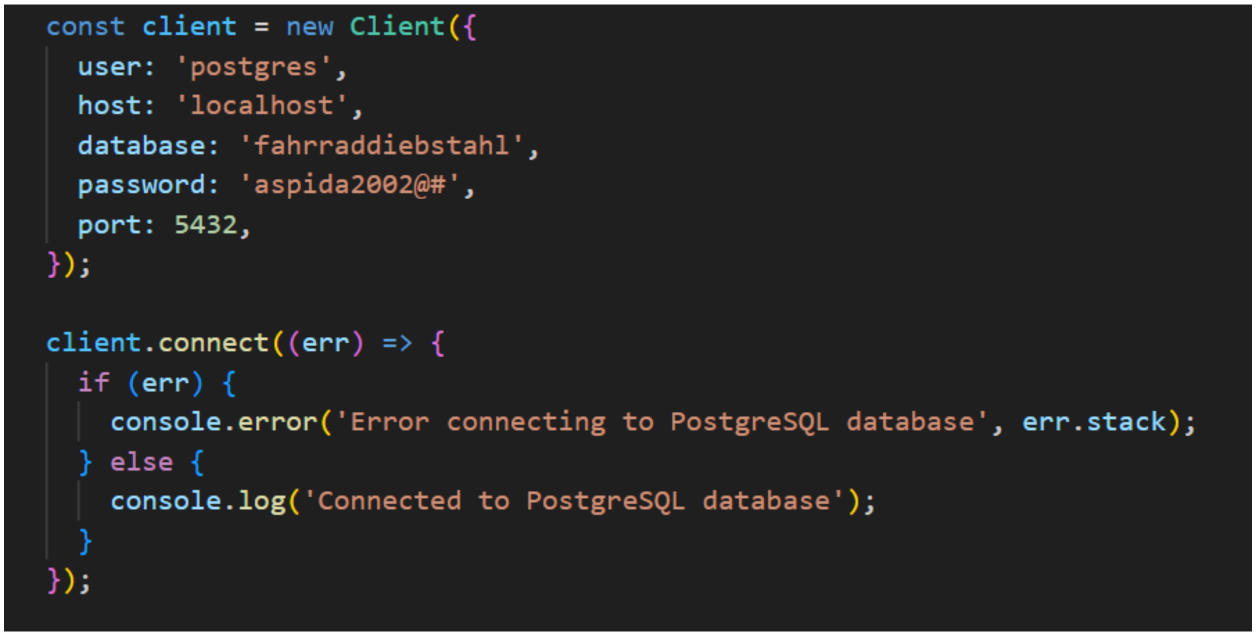
Step 2: Create communication between User Interface and Server
Step 3: Requesting Data from Server through https requests



```
async function onPieChartClickHandler() {
  await axios
    .get('http://localhost:8887/piechart')
    .then((response) => {
      const data = response.data;
      // console.log(response.data);
      // console.log('Server response: ' + Results);
      const results = [];
      for (const key in data) {
        const result = {
          bezirk: data[key].bezirk,
          damage: data[key].schadenshoehe_pro_bezirk,
        };

        results.push(result);
      }
      setResults(results);
```

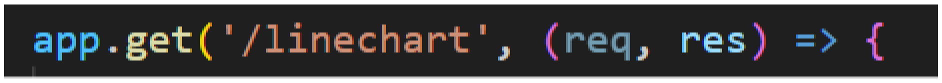
BACKEND:
Technologies: Express JS

Step 1: Create a connection with the database



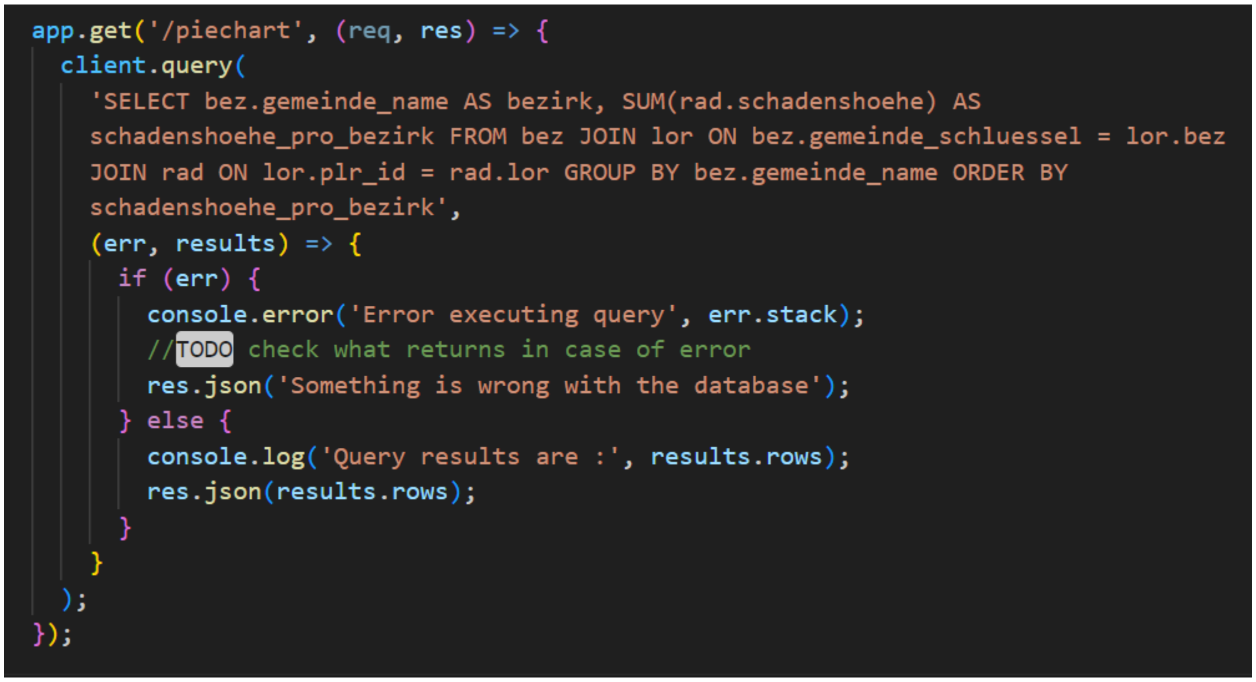
```
const client = new Client({
  user: 'postgres',
  host: 'localhost',
  database: 'fahrraddiebstahl',
  password: 'aspida2002@#',
  port: 5432,
});

client.connect((err) => {
  if (err) {
    console.error('Error connecting to PostgreSQL database', err.stack);
  } else {
    console.log('Connected to PostgreSQL database');
  }
});
```

Step 2: Create endpoints to communicate with frontend



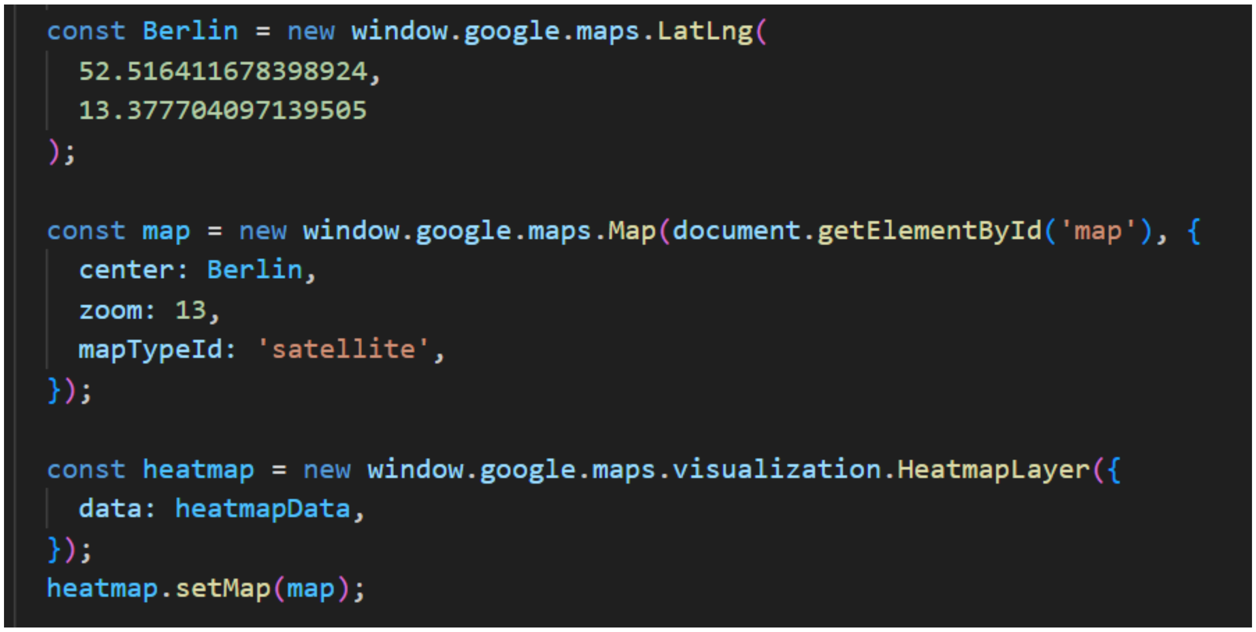
```
app.get('/linechart', (req, res) => {
```

Step 3: Program APIs to retrieve the data we want from database and send them to frontend



```
app.get('/piechart', (req, res) => {
  client.query(
    'SELECT bez.gemeinde_name AS bezirk, SUM(rad.schadenshoehe) AS
    schadenshoehe_pro_bezirk FROM bez JOIN lor ON bez.gemeinde_schluessel = lor.bez
    JOIN rad ON lor.plr_id = rad.lor GROUP BY bez.gemeinde_name ORDER BY
    schadenshoehe_pro_bezirk',
    (err, results) => {
      if (err) {
        console.error('Error executing query', err.stack);
        //TODO check what returns in case of error
        res.json('Something is wrong with the database');
      } else {
        console.log('Query results are :', results.rows);
        res.json(results.rows);
      }
    }
  );
});
```

GOOGLE APIs



```
const Berlin = new window.google.maps.LatLng(
  52.516411678398924,
  13.377704097139505
);

const map = new window.google.maps.Map(document.getElementById('map'), {
  center: Berlin,
  zoom: 13,
  mapTypeId: 'satellite',
});

const heatmap = new window.google.maps.visualization.HeatmapLayer({
  data: heatmapData,
});
heatmap.setMap(map);
```

CHALLENGES:

- Learning new frameworks
- Time management

GITHUB LINK:
https://github.com/oduwancheekee/DBS_project