|  |
| --- |
| OTH Regensburg |
| Spatial Databases |
| Project Report |

|  |
| --- |
| Birgit Huber, Florian Fritz, Florian Fußeder  2.1.2017 |

Inhaltsverzeichnis

[Project Summary 2](#_Toc471214939)

[Decisions 2](#_Toc471214940)

[Questions for the User 3](#_Toc471214941)

[Processing the Data 3](#_Toc471214942)

[Neighborhood-Tabulation-Areas 3](#_Toc471214943)

[School Points 3](#_Toc471214944)

[Colleges and Universities 4](#_Toc471214945)

[Parking lot areas 4](#_Toc471214946)

[Population 4](#_Toc471214947)

[Complaint data 5](#_Toc471214948)

[Rental Data 5](#_Toc471214949)

[Subways 6](#_Toc471214950)

[Soccer fields 6](#_Toc471214951)

[Play areas 7](#_Toc471214952)

[Parks 7](#_Toc471214953)

[Restaurants 8](#_Toc471214954)

[Developing the web application 8](#_Toc471214955)

[Frontend 8](#_Toc471214956)

[General Website Structure 9](#_Toc471214957)

[GeoServer Setup 9](#_Toc471214958)

[Map Display 10](#_Toc471214959)

[Locations of Interest 10](#_Toc471214960)

[Rating 12](#_Toc471214961)

[Showing the data in the map 12](#_Toc471214962)

# Project Summary

The goal of the project was to create a web application that helps a user to find a suitable region for accommodation in a city. By answering a series of questions about the preferences and living circumstances we aim to suggest a ranked list of the zones of a city.

Furthermore, the user should be able to enter some main locations. With the given data we are able to calculate a time- and cost-efficient route.

After some research we choose the city “New York” and discussed the which datasets we might use. Using the datasets, we defined our questions for the user and started with processing the data for an easy integration in our project. Afterwards we were able to start with the development of the final webpage.

# Decisions

We started off by considering 9 which we thought might be interesting and might have good datasets. Each member of the team had a closer look at the city and its availed datasets as following.

* Birgit: San Francisco, Beijing, London
* Florian Fr: Berlin, New York, Tokyo
* Florian Fu: Munich, Seoul, Singapore

After a week we discussed our results and choose New York. One of the main reasons for that is the fact that the United States have a lot of open geo datasets because of an open data initiative[[1]](#footnote-1) . New York City has its own website with data[[2]](#footnote-2) which we used to get the data for our project. New York is by default divided into 5 Zones with total area of approx. 790 km² and 194 Neighborhood-Tabulation-Areas (NTA) with an average size of four km² which perfectly suited the needs of our project. With the questions, we wanted to ask the user, we chose to use the following datasets.

|  |  |
| --- | --- |
| * Parks * Play areas * Restaurants * Soccer Fields * School Points * Parking lots | * Rental Prices * Colleges and University’s * Population * Complaint Data * Subways |

# Questions for the User

Resulting out of our chosen datasets we picked the following questions to determine a suitable NTA for the user.

|  |  |
| --- | --- |
| * Age range * Has Children * Is a student * Owns a car * Has a dog * Does outdoor sports | * Uses subway * Likes nature * Prefers vibrant or quiet areas * Importance of low rental prices * Prefer to live central * Favors specific zones |

# Processing the Data

Each Dataset was processed individually. The result of each dataset a database view containing a rating value between zero to one for each region in terms of the data. For example, the area with the least parking lot area scores the lowest rating, the area with the most scores one. This allows us to easily weight the different ratings in the web application as they are now all in the same range.

## Neighborhood-Tabulation-Areas

As mentioned we chose the NTA areas as foundation of our ratings and zones which we want to suggest to our user. The NTA could easily be imported from the exiting shapefiles[[3]](#endnote-1). The Shapefile contained a NTA code, the NTA name, the geometry and some other metadata. Nothing had to be processed in this table.

## School Points

The School point data[[4]](#endnote-2) was imported as shapefile containing locations as geometry based on the official address. It also includes some basic school information such as Name, Address, Principal, and Principal’s contact information.

The rating of how good a NTA is, in terms of School Points, was determined by the amount of points within the area. This was done by using basic postgis queries[[5]](#endnote-3) ,including ST\_AREA and ST\_CONTAINS. As final step a normalized view which only contained NTA code and the rating[[6]](#endnote-4).

## Colleges and Universities

Like the School points table, the data that we used for the colleges where also easy to import because of the shapefile[[7]](#endnote-5) format, and included some metadata like the name and the street name and the geometry.

Again, the rating of areas containing a college or a university is higher than regions that do not. After viewing the data in the web application we came to the conclusion that because of the low number and the bulked locations only a few areas would have a good rating. So we decided it would be better for our rating to use ST\_DWITHIN and add a distance of 200 for our rating view[[8]](#endnote-6).

## Parking lot areas

The parking lot data also already included the geometry of the area[[9]](#endnote-7). So we just had to prepare the data for our rating.

In this case we have two polygon geometries and we want the area of all parking lots so we used ST\_INTERSECTS. To get an accurate rating that is not biased towards bigger NTA areas, we calculated the m² of parking space per m² in the area[[10]](#endnote-8).

## Population

The population data consisted of a simple csv-file[[11]](#endnote-9) that consisted the data from 2000 and the data of the year 2010.

After removing the data from 2000 we had usable data that only had to be match by NTA-code[[12]](#endnote-10). To get a more accurate rating of how crowded the regions are, we calculated the population per m² in the area.

## Complaint data

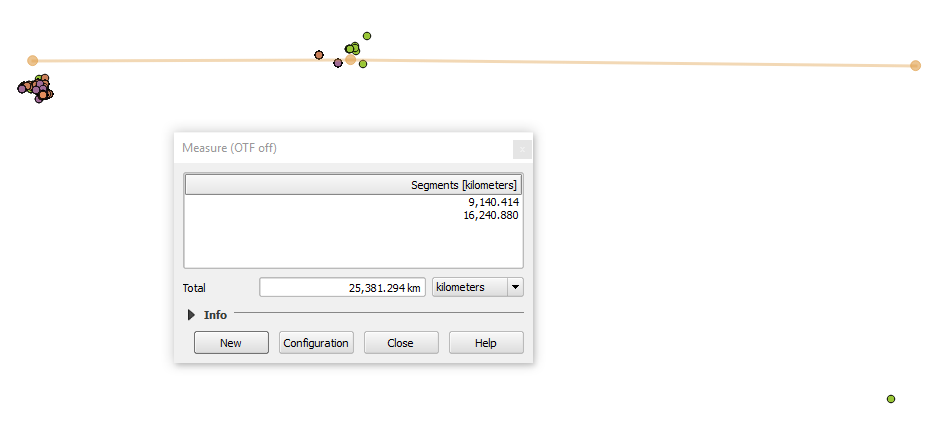
This dataset[[13]](#endnote-11) includes crimes that were reported to the Police for the first three quarters of 2016. The csv-file had the latitude and longitude data some metadata which we cut out to speed up the database queries.

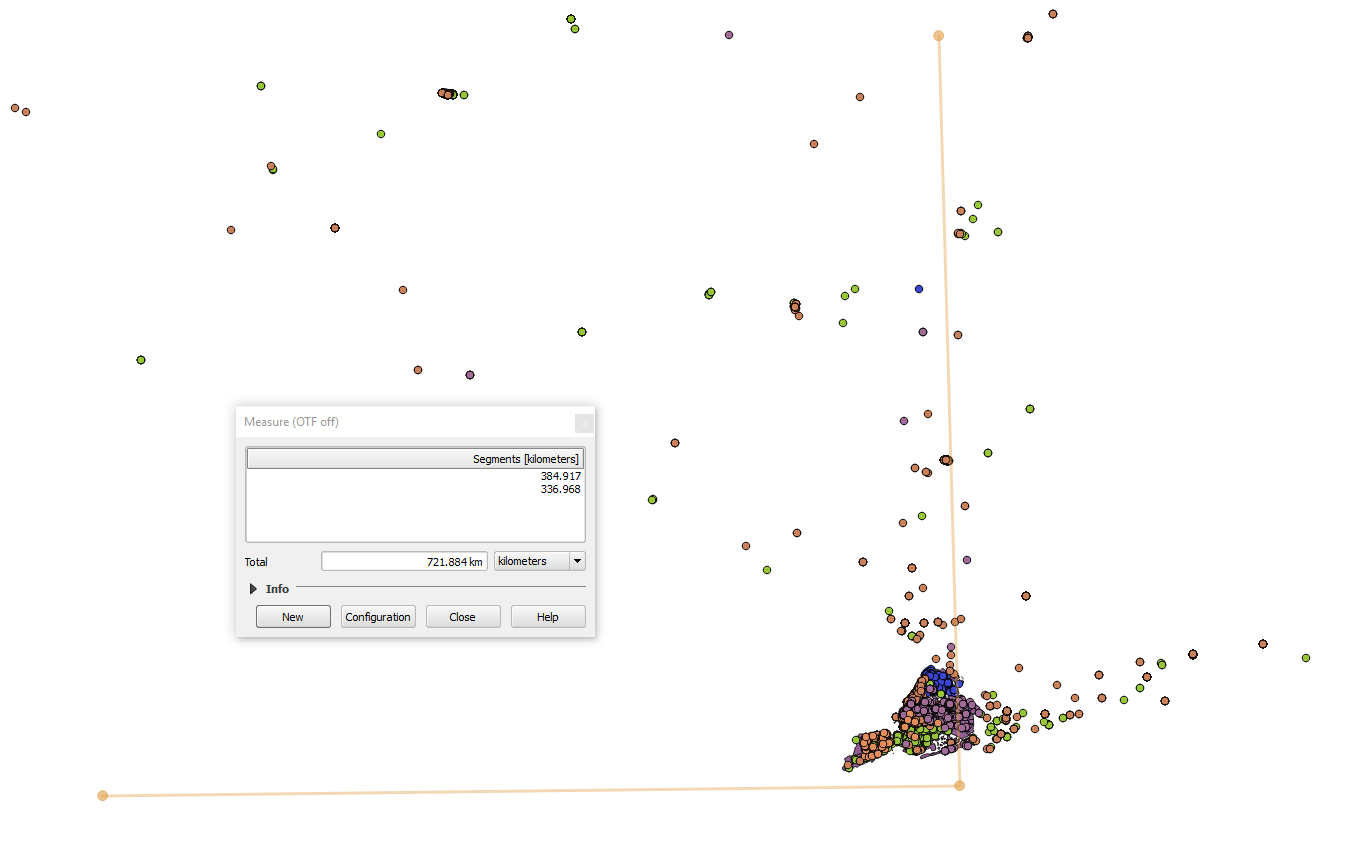
To get a rating of how many crimes have been reported to the police we had to create a point geometry out of the longitude and longitude by using ST\_MAKEPOINT. After that we used ST\_CONTAINS to determine the count of crimes within an area[[14]](#endnote-12).

## Rental Data

To determine a approx. rating of the buying/rental data of the specific areas we used the condominium provided by the NYC department of finances[[15]](#endnote-13).

The datasets consist out of complex Excel Worksheets with much data that where not needed for our project and had to be converted into simple csv-format. Because the data had no spatial data nor a reliable source to match the csv-entries to a specific NTA-area but the address of the building, we decided to use a geocoding[[16]](#footnote-3) API. In our case we used the Python Mapbox API to write a simple script[[17]](#endnote-14) consisting of reading the csv-file, sending the address, receiving the latitude longitude and writing this data into the csv-format. After reviewing the locations of the points that we had produced with the locations from the Mapbox geocoder we discovered that a lot of the produced points where located all over the world. After checking the points, we discovered that approx. 2450 out of the total 23080 entries are clearly out of the boundaries of New York.





Because we decided that we did not want to use such unreliable data, we wrote another Python script[[18]](#endnote-15), but this time we did chose the geocoder API from Google Maps. After checking the locations of the points we created out of the Google latitude and longitude and checking this data in the same way as the Mapbox locations we discovered that only four points where not located in New York. So we did use Google locations to determine average market value per square feet which should give a basic overview of the price range of a NTA.

## Subways

The data for the subway was imported as a shapefile[[19]](#endnote-16) which includes the name of a certain subway station, it’s geometry as a point and other meta data.

Then a new table was created containing the distance to the next subway entrance for each NTA zone. This was done by using the postgis ST\_DISTANCE method we have learned during our lectures. At last there was added a rating to the table which represents the normalised distance for each neighbourhood, with ranges from one if there is a subway entrance in the zone to 0 for the zone furthest away from any subway connection[[20]](#endnote-17). Resulting there was a table containing only the id of the NTA and the rating[[21]](#endnote-18) for further processing.

## Soccer fields

The Soccer field data set was imported as a shapefile[[22]](#endnote-19) into the postgis database program.

The data relation consists of the geometry of the field as well as of its id and other meta data. Similar to the previously described data processing of the subway, for each NTA area was calculated the distance to the closest football field[[23]](#endnote-20), as there are not that many field in New York. So in the end there was a relation with the id of the NTA and its normalised distance to the next soccer field[[24]](#endnote-21).

## Play areas

The data for the play areas were also imported as a shapefile from the official geo data website of New York City[[25]](#endnote-22). It included the geometry of the park and other meta data which we were of no use for us.

First the number of playgrounds in a certain NTA region was calculated by joining the NTA with the playareas table on whether a certain play ground is in that NTA zone using ST\_INTERSECTS. As a further step the same procedure was repeated to find play areas near the zone using ST\_DISTANCE with a distance of 500m, as it seemed reasonable to also considerer those parks. Afterwards those two parts were put together on rating the parks in the NTA area thrice the weight of the ones in the adjacent region[[26]](#endnote-23). Finally, the rating was calculated by considering the total amount of playgrounds in and nearby the zone as well as the zone’s area[[27]](#endnote-24). Those steps were all performed successively as of a lack of performance of the database.

## Parks

The data for the parks and recreational areas[[28]](#endnote-25) of New York City was also downloaded as a shapefile and imported into the postgis database program.

The table consists of several columns including the geometry, location, name and other attributes. Firstly, the total area of all parks in a particular NTA zone was calculated by joining the previously imported park table with the NTA one. Therefor the postgis functions ST\_AREA, ST\_INTERSECTS and ST\_INTERSCETION which were presented during our lectures were used. As in this case not only the parks in a certain NTA region, but also the ones close by seem to be relevant, a buffer of 500 m was added around the zone to also determine the possible adjacent recreational areas. Afterwards the total area of all parks counting for a certain NTA region were summed up and divided through the area of the NTA zone to also take that under consideration. For this calculations the database functions ST\_BUFFER, ST\_SETSRID and ST\_DIFFERENCE were used[[29]](#endnote-26). In the end this number was normalised as the rating with a range from zero to one regarding how much park area exists in this NTA neighbourhood[[30]](#endnote-27). These procedures were all executed in small steps as of the performance of the database program.

## Restaurants

To get the data for the restaurants a table[[31]](#endnote-28) was downloaded as a CSV file from the official governmental geo data website of New York City and imported as a text file into the database program. That table included an address with a street name and zip code of a restaurant and several other meta data.

As for further processing a geometry data entry was needed to determine the exact location of the restaurant a second data set[[32]](#endnote-29) which matched addresses with a geometry was imported. So, for further processing of that data, irrelevant information like building numbers and other meta data not needed were removed that it contains only valid data with a geometry object and every data only once. After eliminating multiple entries to one restaurant in the restaurant table the two relations were join together using the address with street and zip code. However, this only worked in about half of the cases and the rest could not be matched as of differences in the street name as sometimes there were additionally building numbers in the name or abbreviations were used. As there did not seem to be a better solution the Google Geocoding API was used. For that the not matched restaurants were extracted into another table and exported as text files.

A Java program[[33]](#endnote-30) was developed to read the text files, send for each entry a request to the API which returns the latitude and longitude of the address and save the results in another file. An issue of the API was that the necessity of using a key for the request and the validation of the key which was at about 2500 requests. To avoid this more keys could be requested from the Google API site for other projects. With that solution only about three percentage of the restaurants could not be matched to a location and have to be omitted for further processing.

Afterwards the file with the geo coded data was imported and added to the table with the other restaurants. Finally, the NTA zone’s table was joined with the restaurant’s using the ST\_INTERSECTS function to get the total number of restaurants for each NTA neighbourhood[[34]](#endnote-31). Resulting, the id of the NTA zones and the rating of the total number of restaurants in the particular area were used for further processing[[35]](#endnote-32).

# Developing the web application

## Frontend

## General Website Structure

We did not use a backend application sever to keep the website setup and structure simple. The whole website is built as a single html page that executes all code necessary to display the final map on the client’s browser. The different steps and sections on the page are implemented by hiding and revealing parts of the web page.

We used various libraries for the website. The main ones are the JavaScript library JQuery for working with the HTML elements on the page and the SemanticUI CSS framework to style the page without much custom CSS code.

One thing we could see when building the application is that modern website building tools, like webpack[[36]](#endnote-33) or gulp[[37]](#endnote-34), and a backend application sever should be used if the project was any bigger or more complex.

## GeoServer Setup

We used GeoServer to access our map data. First we joined all individual ratings that were created into one table. This table contains all zones with their metadata, geometry and ratings. We decided to use a table and not a view at this point, as this acts as our caching layer. The table is created once, which is time consuming, as all spatial operations to calculate the ratings have to be evaluated. Accessing the table later on is very fast.

After that we setup a GeoServer and connected it to the PostGIS database containing the table. We then setup a layer that allows us to access the table from our web page code. One challenge at this point was the same-origin policy[[38]](#endnote-35), which prevents the clients JavaScript code from loading JSON data from the GeoServer.

To solve the issue one can either setup the GeoServers configuration to allow cross-origin resource sharing (CORS) or make sure that the GeoServer's IP and port are the same as the webpages. We decided to use the second method, as the GeoServer configuration is not trivial and makes it harder to setup the project. To get the websites ip and port to be the same as the GeoServer ones we setup a webserver that delivers our static website and also acts as a proxy for the GeoServer. To do so we used a npm tool called 'local-web-server', as it is very easy to install and use. After installing it we can start it using the command `ws --rewrite '/geoserver/\* -> http://localhost:8080/geoserver/$1'`. Note the --rewrite option. It proxies all requests made to '/geoserver/' to the actual GeoServer[[39]](#endnote-36). We recommend to look at the 'local-web-server' tool for any work with small web projects, as it is also very useful to simply serve some local files to the browser.

## Map Display

With the GeoServer setup we simply used a JQuery ajax request to load the maps data as GeoJSON from the GeoServer. We decided to load the map as GeoJSON, as it allows us to build more interactive content, e.g. style the map dynamically, and as it gives us access to all the rating values that are included in the table setup above. To request the map data as GeoJSON instead from the GeoServer we had to add '&outputFormat=application%2Fjson' to the request that accesses the data[[40]](#endnote-37).

After this data has been loaded from the server it can be used to display the different zones on the map. Colours and styling of the zones are based on a computed rating value, more on that it TODO: ref chapter with ratings.

Besides, that most map setup and interaction is mostly cosmetic, like restricting the map are to New York, clicking interactions and hover effects.

## Locations of Interest

On the second question step the user can input a route that he typically does daily. The route always starts and ends at the centre of a zone. This route is then used to calculate the walking distance for each region. To implement this, we had to do two main steps. First we had to let the user input his locations of interest, then we had to use these inputs to calculate walking distances.

To let the user input an arbitrary number of locations we used JQuery, SemanticUI search and some custom data structures. To execute the actual search for a specific location of interest we implemented some of I search methods. In this method we did use the geocoding API of mapbox to get a suggestion of locations based on the user’s current input. The auto completion is done by SemanticUI. After the user has selected the location he was looking for we use the coordinates given by the mapbox geocoding to display a marker on a small map besides the search input. This helps the user to visually see what he selected. We also save all selected places in an array for later use.

After the user finished the input we need to get the walking distances for the given route. To do this we need to calculate this distance starting and ending at each of the 194 zones centres, as we want to see how long the route takes for each individual zone. Because of the amount of zones, we could not use the normal directions API by mapbox, as this would result in 194 requests, which takes too long and also exceeds the API limit of mapbox[[41]](#endnote-38). To solve this, we used the mapbox distance API, which is currently in preview[[42]](#endnote-39). The API allows to send an array of up to 100 GPS locations and responds with a matrix containing the walking distances between all the points. The matrix describes a graph with weighted edges based on the walking distances.

In our use case we send the users inputted places in the centres of the zones in batches to mapbox, as we could not send all 194 zones at once. To coordinate these asynchronous requests, we used promises. We then used this matrix to calculate the routes starting/ending in each individual zone centre and saved the results from the requests into the feature properties of each zone. We then normalise the values, so they are between zero and one. This allows us to use the dynamically calculated walking distances just like all static ratings from the GeoServer.

The distance API worked really well for us and we would absolutely recommend it for all problems where walking distances between many points are needed. Note that at the time of writing you have to send a request to the support to access the distance API, otherwise all requests will fail with an error code.

## Preference Preselection

At the beginning of the application the user is asked to answer a few question that allows us to select an appropriate NTA zone for him. For that we determined a few peer groups, like family with children and dogs or student, who may have different preferences in selecting their living environment.

So for instance we thought that a student might like to live in a vibrant and central area not too far away from a university but not pay too much. In comparison the family may prefer living in a quieter area close by schools and playgrounds for the children and parks for the dog.

Altogether depending on the user’s selections an importance factor to is assigned to a certain point and based on that the best NTA zoned are ranked. Furthermore, the user can adjust the exact values of the questions as he prefers later on.

## Rating

Whenever any user input changes we trigger a rating function that calculates a numeric valuation value for each zone. We then use these valuations to colour the zones and display a sorted list of results in the sidebar. This way we and the user can change the rating inputs at any time and the changes are displayed on the user interface.

For that we have added adjustment sliders for how important a short distance to the locations of interests, the parking situation, cheap rental prices, playgrounds, parks and recreational areas as well as restaurants or subway stations nearby are.

Furthermore, the user can select whether he prefers living downtown or suburban, in a quiet or vibrant area or next to a school, university or playground or not.

So depending on the importance of the criteria points are added to a NTA zone. This is calculated by the factor the user selects with the sliding bar, ranging from -1, when a certain point is unwanted or zero, when the question is not something the users cares about to one which is absolutely necessary, and the rating of the NTA zone in this point. Additionally, we added programmatically weight factors to some points we thought more important. So for example the short distance to the user’s locations of interest and the subway situation are considered very important so a higher weight is multiplied to the result.

As we discovered that the rating of the subway shows generally very high values we wanted to rate down NTA zones with no subway stations. So we lowered the rating value for NTAs with no access to the subway.

1. https://www.data.gov/ [↑](#footnote-ref-1)
2. https://nycopendata.socrata.com/ [↑](#footnote-ref-2)
3. <https://data.cityofnewyork.us/City-Government/Neighborhood-Tabulation-Areas/cpf4-rkhq> [↑](#endnote-ref-1)
4. <https://data.cityofnewyork.us/Education/School-Point-Locations/jfju-ynrr/data> [↑](#endnote-ref-2)
5. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/4.publicSchoolPoints/SchoolData.sql> [↑](#endnote-ref-3)
6. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/4.publicSchoolPoints/view.txt> [↑](#endnote-ref-4)
7. <https://data.cityofnewyork.us/Education/Colleges-and-Universities/4kym-4xw5> [↑](#endnote-ref-5)
8. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/2.colleguesAndUniversitys/view.txt> [↑](#endnote-ref-6)
9. <https://data.cityofnewyork.us/City-Government/Parking-Lot/h7zy-iq3d> [↑](#endnote-ref-7)
10. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/3.parkintLot/ParkingData.sql> [↑](#endnote-ref-8)
11. <https://data.cityofnewyork.us/City-Government/New-York-City-Population-By-Neighborhood-Tabulatio/swpk-hqdp> [↑](#endnote-ref-9)
12. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/6.population/populationData.sql> [↑](#endnote-ref-10)
13. <https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Current-YTD/5uac-w243> [↑](#endnote-ref-11)
14. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/FuFlo%20Data/TABLES/7.complaint/complaintData.sql> [↑](#endnote-ref-12)
15. <http://www1.nyc.gov/site/finance/taxes/property-cooperative-and-condominium-comparables.page> [↑](#endnote-ref-13)
16. Computational process of transforming a postal address to a location (latitude longitude) [↑](#footnote-ref-3)
17. <https://github.com/FlorianFusseder/SpatialDatabases/tree/master/Geocoder/Geocoder_mapbox> [↑](#endnote-ref-14)
18. <https://github.com/FlorianFusseder/SpatialDatabases/tree/master/Geocoder/Geocoder_google> [↑](#endnote-ref-15)
19. <https://data.cityofnewyork.us/Transportation/Subway-Stations/arq3-7z49> [↑](#endnote-ref-16)
20. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/subway/subway.sql> [↑](#endnote-ref-17)
21. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/subway/subwaydistance_table.txt> [↑](#endnote-ref-18)
22. <https://data.cityofnewyork.us/Recreation/Map-of-Soccer-and-Football-Fields/qqsi-vm9f> [↑](#endnote-ref-19)
23. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/soccerfields/soccerfields.sql> [↑](#endnote-ref-20)
24. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/soccerfields/soccerfield_data.txt> [↑](#endnote-ref-21)
25. <https://data.cityofnewyork.us/City-Government/Play-Areas/8fhn-c4v3> [↑](#endnote-ref-22)
26. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/playgrounds/playgrounds.sql> [↑](#endnote-ref-23)
27. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/playgrounds/neighbourhood_playground_table2.txt> [↑](#endnote-ref-24)
28. <https://data.cityofnewyork.us/City-Government/Parks-Properties/rjaj-zgq7> [↑](#endnote-ref-25)
29. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/parks/park2.sql> [↑](#endnote-ref-26)
30. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/parks/neighbourhood_parks_table2.txt> [↑](#endnote-ref-27)
31. <https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/xx67-kt59> [↑](#endnote-ref-28)
32. <https://data.cityofnewyork.us/City-Government/NYC-Address-Points/g6pj-hd8k> [↑](#endnote-ref-29)
33. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/restaurants/geoCoding.java> [↑](#endnote-ref-30)
34. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/restaurants/restaurants2.sql> [↑](#endnote-ref-31)
35. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/data/restaurants/restaurants_table2.txt> [↑](#endnote-ref-32)
36. <https://webpack.github.io/> [↑](#endnote-ref-33)
37. <http://gulpjs.com/> [↑](#endnote-ref-34)
38. <https://en.wikipedia.org/wiki/Same-origin_policy> [↑](#endnote-ref-35)
39. <https://github.com/FlorianFusseder/SpatialDatabases/tree/master/frontend#setup-geoserver> [↑](#endnote-ref-36)
40. <https://github.com/FlorianFusseder/SpatialDatabases/blob/master/frontend/script.js#L51> [↑](#endnote-ref-37)
41. <https://www.mapbox.com/api-documentation/#directions> [↑](#endnote-ref-38)
42. <https://www.mapbox.com/api-documentation/#distance> [↑](#endnote-ref-39)