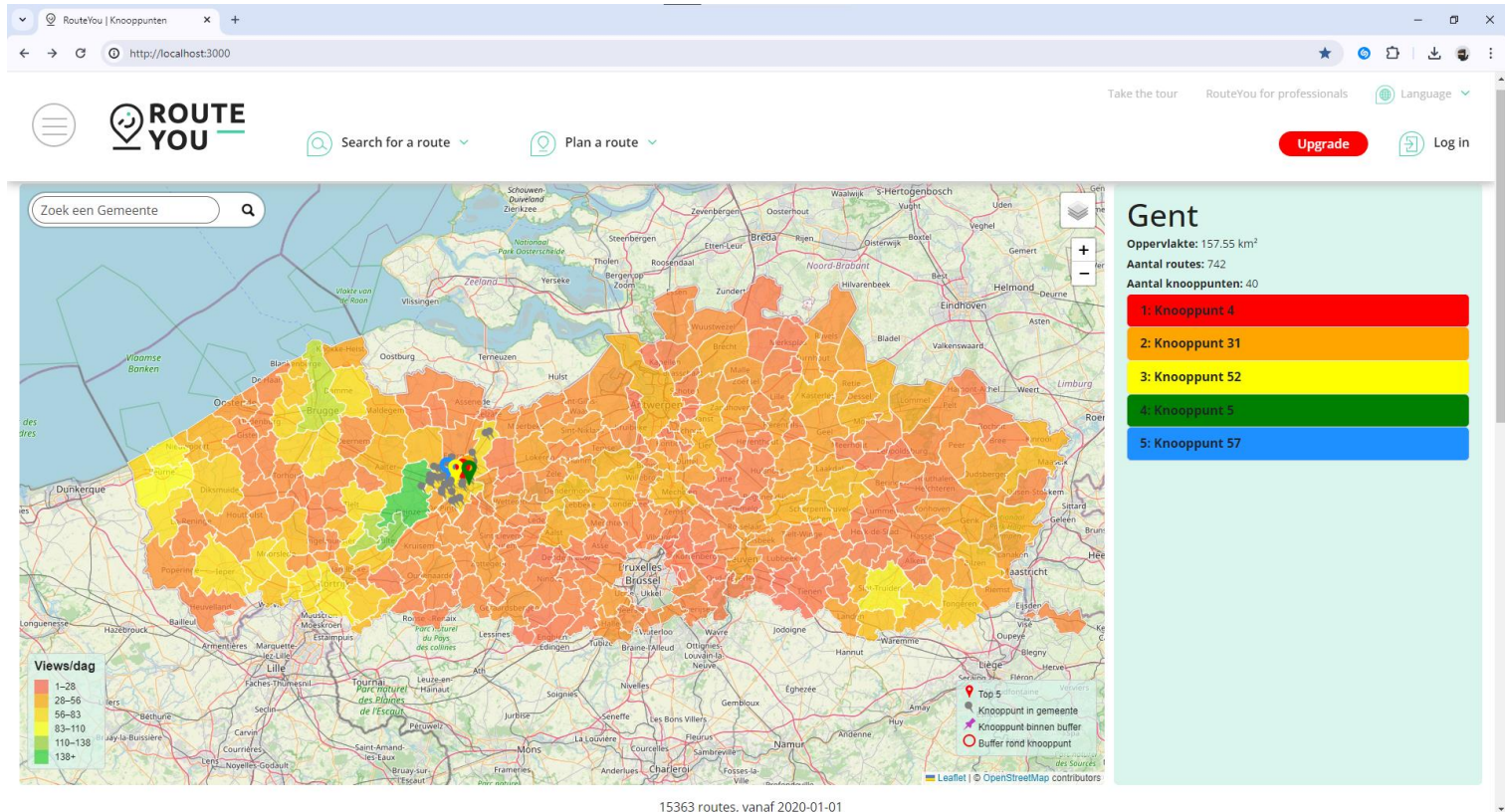


Webapplicatie fietsknooppuntpopulariteit Vlaanderen: Technische Handleiding V1.2

Door Ferre Verstichelen, 2024



Inhoud

Benodigde Software:.....	4
Installatie van software:	4
Visual Studio Code	4
Node.js	4
PostgreSQL	5
PostGIS.....	6
Modules en Packages installeren.....	9
Hoe voer ik dit zelf uit?	10
A. Download en installeer de nodige software. (zie deze stap)	10
B. Download routes (indien nog niet beschikbaar).	10
C. Verwerk de routes. (zie deze stap)	10
D. Start de webapplicatie. (zie deze stap)	10
1. Data verzamelen	10
2. Data voorbereiden.....	11
1. 1getRefGem.js	12
2. 2gemeentesJSONtoTABLE.js	13
3. 3knooppuntenJSONtoTABLE.js	14
4. 4DBtoGEOJSON.js	16
3. Structuur Databank	17
gemeente	17
knooppunt.....	17
feature	18
Koppeling tabellen	18
gemeente – knooppunt	18
gemeente – feature	18
knooppunt – feature	18
Extra queries.....	18
save.js	18
dropTables.js.....	19
4. Webapplicatie opzetten	20
app.js	20
Scripts gebruikt door app.js.....	21
1. /fetchNetwerk.js	21
2. /getGemeenteTop5.....	21
3. /getGemNotTop5:.....	22

4. /getFromRadius	24
index.html	26
1. html	26
2. JavaScript.....	27
404.html.....	29
Veelgebruikte bronnen.....	30

Benodigde Software:

- Visual Studio Code / gelijkaardige software
- Node.js
- PostgreSQL + PostGIS extensie

Installatie van software:

Visual Studio Code

Download de laatste stabiele versie van Visual Studio Code voor jouw besturingssysteem [hier](#) en doorloop de installatiewizard.

Node.js

Download de laatste “LTS” versie voor jouw besturingssysteem [hier](#) en doorloop de installatiewizard.

Download Node.js®

Download Node.js the way you want.

[Prebuilt Installer](#) [Prebuilt Binaries](#) [Package Manager](#) [Source Code](#)

I want the


v20.12.1 (LTS)

 version of Node.js for

Windows

 running

x64

 **Download Node.js v20.12.1**

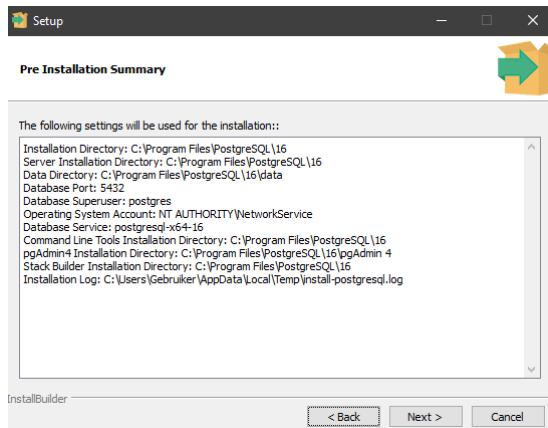
1. Als alles goed ging, kan je nu je CMD openen en “node -v” uitvoeren om te zien of node actief is, en of je de gewenste versie hebt.

```
C:\> Opdrachtprompt
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. Alle rechten voorbehouden.

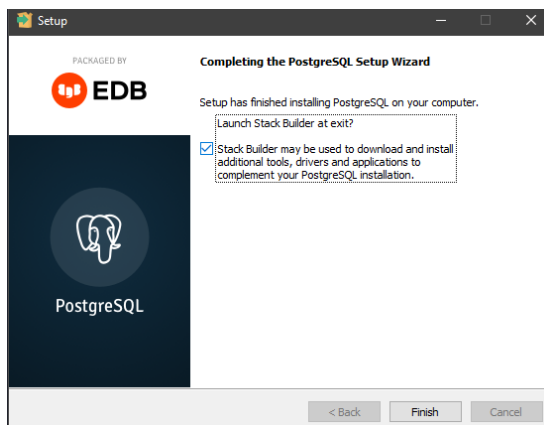
C:\Users\Gebruiker>node -v
v21.7.1
```

PostgreSQL

1. Download de laatste versie van PostgreSQL [hier](#)
2. Begin de installatie, laat de “Installation Directory” ongewijzigd.
3. Laat bij “Select Components” ook alles ongewijzigd.
4. Laat “Data Directory” ook ongewijzigd.
5. Bij “Password” vul je 2 maal hetzelfde wachtwoord in. Om later geen code te moeten veranderen kies je voor wachtwoord “root”.
6. Bij “Port” laat je de “5432” staan en ga je verder.
7. Bij “Advanced Options” kies je wat je wil.
8. Nu krijg je de “Pre-Installation Summary”, klik op NEXT.

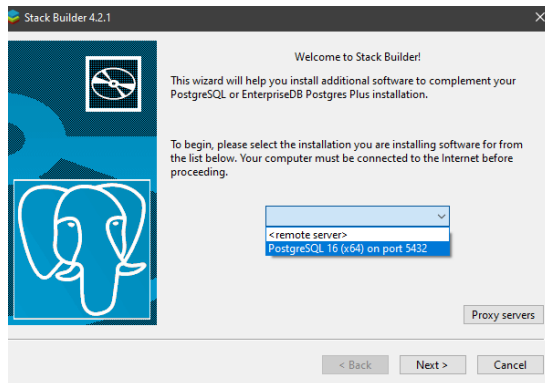


9. Klik nogmaals op NEXT en wacht tot de installatie voltooid is.
10. Na de installatie van PostgreSQL kan je de PostGIS extensie installeren. Laat daarvoor de optie “Launch Stack Builder” aangevinkt na het voltooien van de Installatie. Klik op FINISH

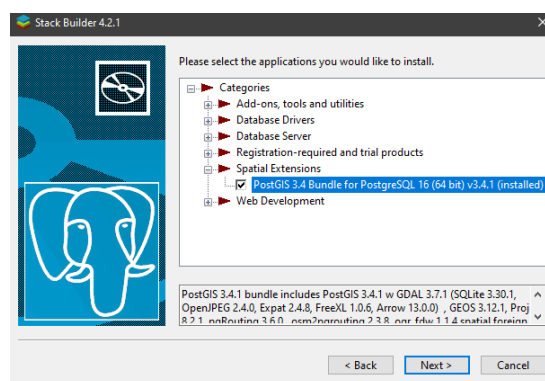


PostGIS

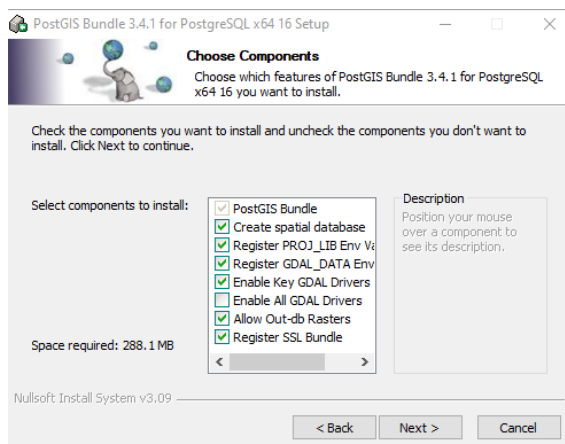
1. Kies in de Stack Builder jouw PostgreSQL installatie. Klik op NEXT.



2. Vouw “Spatial Extensions” uit en vink “PostGIS” aan. Klik terug op NEXT.

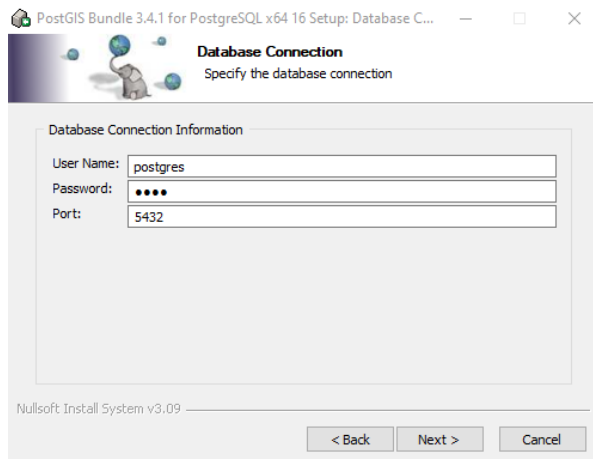


3. Laat “Download Directory” ongewijzigd en klik op NEXT.
4. Klik nogmaals op NEXT om de installatie van PostGIS te starten.
5. Accepteer de Gebruiksvoorwaarden.
6. Bij “Choose Components” vink je zeker “Create spatial database” aan. Klik op NEXT.

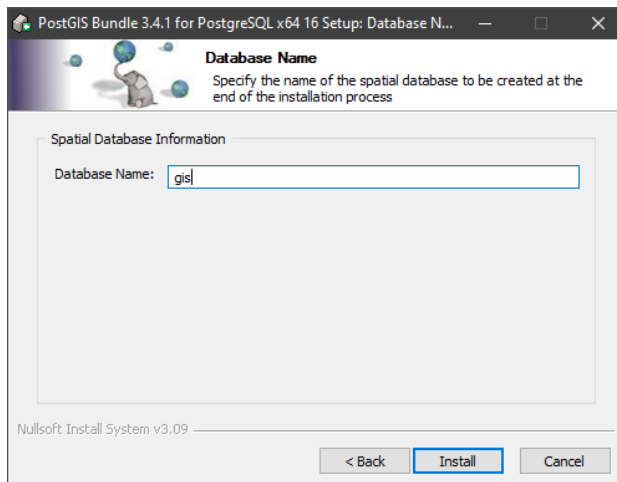


7. Laat de Installatielocatie ongewijzigd en klik op NEXT.

8. Bij “Database Connection” vul je bij Password “root” in. Klik op NEXT.

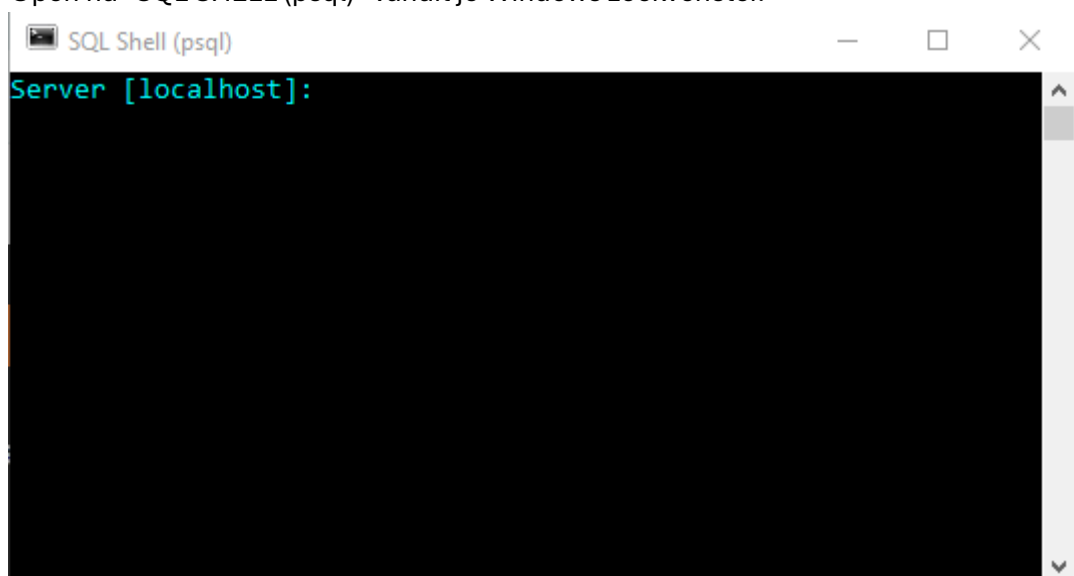


9. Bij “Database Name” vul je “gis” in. Klik op INSTALL.



10. Na de installatie klik je op FINISH.

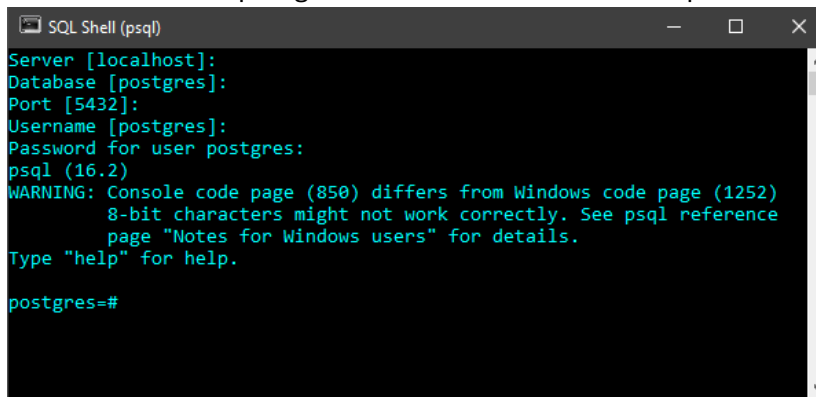
11. Open nu “SQL SHELL (psql)” vanuit je Windows zoekvenster.



12. Server: Localhost: niets invoeren, druk op ENTER.

13. Database: Postgres: niets invoeren, druk op ENTER.

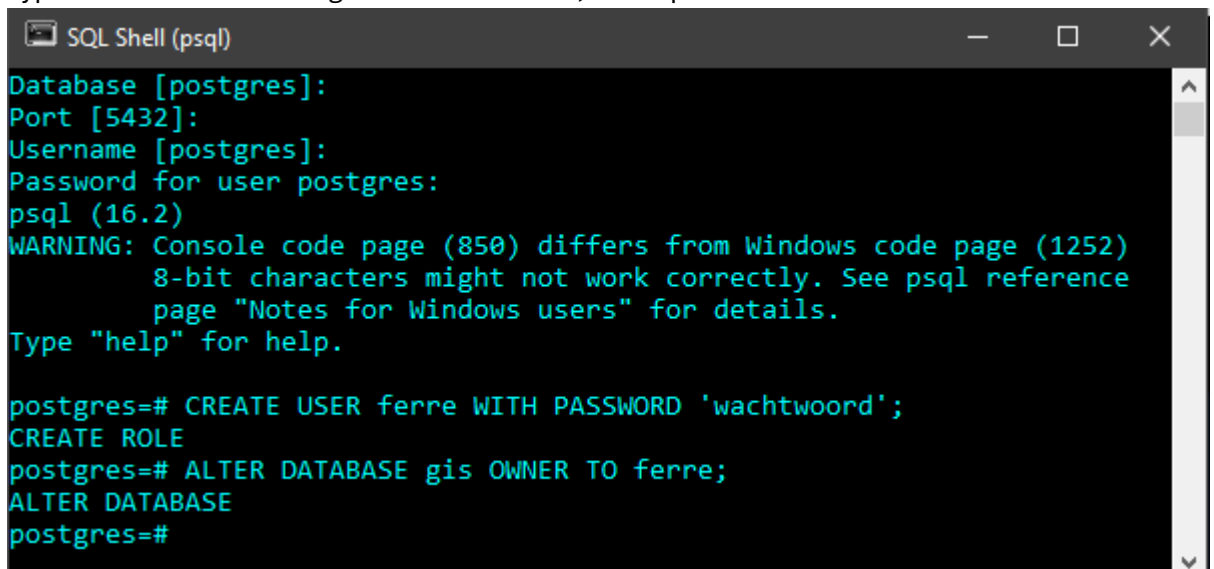
14. Port: 5432: niets invoeren, druk op ENTER.
15. Username: postgres: niets invoeren, druk op ENTER.
16. Password for user postgres: vul hier “root” in en druk op ENTER.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.2)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

17. Je bent nu ingelogd, je kan nu postgres verder configureren. We maken nu onze gebruiker aan.
18. Typ nu “*CREATE USER ferre WITH PASSWORD 'wachtwoord';*” klik op ENTER.
19. Typ nu “*ALTER DATABASE gis OWNER TO ferre;*” klik op ENTER.



```
SQL Shell (psql)
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.2)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

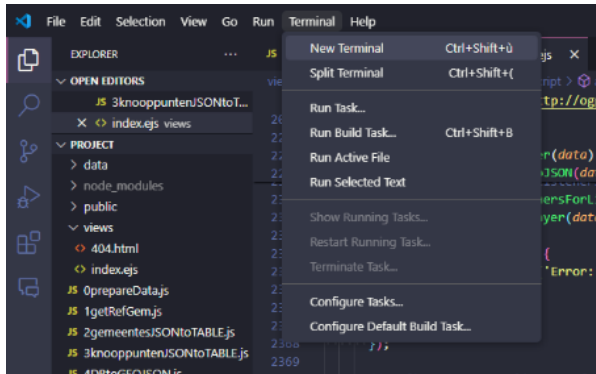
postgres=# CREATE USER ferre WITH PASSWORD 'wachtwoord';
CREATE ROLE
postgres=# ALTER DATABASE gis OWNER TO ferre;
ALTER DATABASE
postgres=#
```

20. Typ nu “\q” en druk TWEEEMAAL op ENTER om het venster te sluiten.

Modules en Packages installeren

Om de nodige packages te installeren doe je het volgende:

1. Controleer of “package.json” en “package-lock.json” aanwezig zijn in de Projectmap.



2. Voer in de terminal “npm install” uit (Node installeert nu de nodige modules).

```
PS C:\Users\Gebruiker\WERKPLEKLEREN GEO ICT\JavaScript\Project> npm install

added 172 packages, and audited 173 packages in 3s

19 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Gebruiker\WERKPLEKLEREN GEO ICT\JavaScript\Project>
```

Hoe voer ik dit zelf uit?

A. Download en installeer de nodige software. (zie [deze stap](#))

B. Download routes (indien nog niet beschikbaar).

Indien deze nog moeten worden gedownload: stel in hoeveel je er wil en hoe oud deze mogen zijn. (zie [deze stap](#))

C. Verwerk de routes. (zie [deze stap](#))

Voer hier de eerste 4 stappen uit.

D. Start de webapplicatie. (zie [deze stap](#))

Voer de command “nodemon app” uit in de terminal.

1. Data verzamelen

Voor het verzamelen van de data wordt een node.js script genaamd “getRouteDumps.js” gebruikt.

Voordat je het script uitvoert kan je kiezen hoeveel routes je wil downloaden van api.routeyou.com.

Vul jouw keuze in bij “num_route_dumps” op lijn 7 van het script.

```
JS getRouteDumps.js > ...
1  // Required modules
2  const fs = require("fs");
3  const http = require("http");
4  const path = require("path");
5
6  // Specify the number of route dumps to download (will be met approximately)
7  const num_route_dumps = 5000;
8  // Specify the minimum date for route creation (format: YYYY-MM-DD)
9  const minDate = "2023-01-01";
10
```

Dit script zal vervolgens het gewenste aantal routes opvragen bij de RouteYou webservice API. Daarvoor wordt gewerkt met batches van 10, waardoor er (na verdere filtering van de resultaten) altijd wel een paar routes extra worden gedownload dan werd gevraagd (dit zijn er meestal 3 extra).

Als het script voor het eerst wordt uitgevoerd, wordt naast een .txt bestand “last_route_id.txt” met daarin het laatst gedownloade route id en het laatste gebruikte getal voor de naamgeving van de .json bestanden.

Wanneer je het script een tweede maal uitvoert, wordt dit txt-bestand gelezen en zal de naamgeving automatisch verder gezet worden, en zullen enkel route-id's worden gezocht die hoger (en dus recenter) zijn dan de laatst gedownloade route. Zo kan je zonder problemen op verschillende momenten je verzameling routes uitbreiden.

In het geval de routes plots niet meer verder worden gedownload, en het script vast loopt op een bepaalde route, kan dit makkelijk verklaard worden. Op lijn 149 van het script wordt meegegeven vanaf welke datum we routes zoeken. Het is mogelijk dat het script dus vastloopt omdat er geen routes meer zijn die aan alle criteria voldoen.

Er zijn dus 2 belangrijke variabelen in dit script die eventueel kunnen worden aangepast:

- Lijn 7: “num_route_dumps”: kies hoeveel routes je wil downloaden.
- Lijn 9: “minDate”: kies vanaf welke creatiedatum je routes wil downloaden.

Verdere details over dit script kan je vinden in de commentaar bij de code.

```
142 // Main loop to fetch and process route dumps
143 (async () => {
144   while (filesProcessed < num_route_dumps) {
145     // Conditions for route search
146     // indien je meer routes wil ophalen, kan je de
147     // createdAt.min aanpassen naar een vroegere datum
148     const conditions = {
149       "characteristic.id": 43,
150       "id.min": min_route_id,
151       "createdAt.min": minDate,
152       "type.id": [1, 5, 6, 7, 47] // Add the condition for type.id so that only bike routes are fetched
153     };
```

De output van dit script ziet er als volgt uit: (de .json-bestanden zijn te lang om hier een voorbeeld toe te voegen, maar neem gerust een kijkje)

File Explorer	Terminal
data\input	data > input > last_route_id.txt
last_route_id.txt	1 14647169, 15363
route_dump_1.json	public > site_info.txt
route_dump_2.json	1 15363, 2020-01-01
route_dump_3.json	
route_dump_4.json	
route_dump_5.json	
route_dump_6.json	

Bijkomend wordt nog een ander .txt bestand genaamd “site_info.txt” aangemaakt door dit script, dit wordt opgeslagen in de map “public”.

Hierin wordt bijgehouden vanaf welke datum er routes worden gedownload en hoeveel er zijn gedownload. Deze info wordt uiteindelijk rechtstreeks weergegeven op de webpagina.

We downloaden hier 15362 routes omdat dit toevallig het aantal beschikbare routes gemaakt sinds 1 januari 2020 is. Voor het downloaden deze 15.000-tal routes voorzie je best ongeveer een uur.

2. Data voorbereiden

Eenmaal het gewenste aantal routes werden gedownload, zijn deze klaar om te verwerken.

1. Open het project in Visual Studio Code
2. Open het bestand “3knooppuntenJSONtoTABLE.js”
3. Hier kan je kiezen hoeveel routes je wil verwerken (voor het prototype gebruiken we 15362 routes).

```

JS 3knooppuntenJSONtoTABLE.js > ...
1  const fs = require("fs");
2  const { Client } = require("pg");
3  const path = require("path"); // Import the path module for resolving paths
4
5  // KIES HIER HOEVEEL ROUTES JE WIL VERWERKEN
6  const numberOfJsons = 15362;
7  // PostgreSQL connection configuration
8  const pgConfig = {
9    user: "ferre",
10   host: "localhost",
11   database: "gis",
12   password: "wachtwoord",
13   port: 5432, // default port for PostgreSQL
14 };

```

4. Open de **Terminal in Visual Studio Code** en voer “**node 0prepareData**” uit. Deze command voert “**0prepareData.js**” uit, wat op zich weinig doet, maar wel de 4 scripts voor het voorbereiden van de data om de beurt uitvoert als “child-processes”.

```

// Define an array of script paths to run sequentially
const scripts = [
  '1getRefGem.js',
  '2gemeentesJSONtoTABLE.js',
  '3knooppuntenJSONtoTABLE.js',
  '4DBtoGEOJSON.js'
];

```

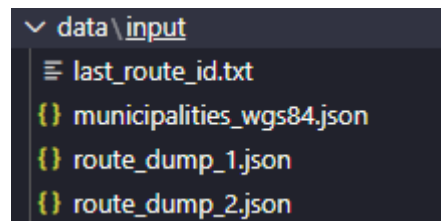
1. 1getRefGem.js

Indien het bestand nog niet bestaat, haalt dit script het meest recente referentiebestand van Vlaamse Gemeentegrenzen op van de API van geo.api.vlaanderen.be. Dit bevat de geometrie van de grenzen van alle Vlaamse gemeentes.

Gezien de Leafletkaart werkt met het WGS84 coördinatenstelsel, vragen we de coördinaten op in dit formaat.

Dit wordt opgeslagen in geojson-formaat, maar met de gewone “.json” bestandsextensie.

Dit bestand wordt in dezelfde map opgeslagen als de routes, onder de naam “municipalities_wgs84.json”.



2. 2gemeentesJSONtoTABLE.js

Dit is het eerste script dat verbinding maakt met de PostGIS databank.

Indien je een andere naam/wachtwoord koos tijdens de installatie van PostGIS, kan je die hier invullen op lijn 6-11.

```
JS 2gemeentesJSONtoTABLE.js > main
1 // Import necessary modules
2 const fs = require('fs'); // Module for file system operations
3 const { Client } = require('pg'); // PostgreSQL client module
4
5 // PostgreSQL connection configuration
6 const pgConfig = {
7   user: 'ferre', // Username for database access
8   host: 'localhost', // Hostname where the database server is running
9   database: 'gis', // Name of the database
10  password: 'wachtwoord', // Password for database access
11  port: 5432, // Port number for PostgreSQL (default is 5432)
12 };
```

Het script kijkt of er al een tabel “gemeente” bestaat, en maakt er één aan als dat nog niet zo is.

```
// Create table if it doesn't exist
await client.query(`
CREATE TABLE IF NOT EXISTS gemeente (
  id SERIAL PRIMARY KEY,
  refgem TEXT,
  naam TEXT,
  viewCount DOUBLE PRECISION DEFAULT 0,
  total_views INT DEFAULT 0,
  route_count INT DEFAULT 0,
  geom geometry(Geometry, 4326),
  bbox_min geometry(POINT, 4326),
  bbox_max geometry(POINT, 4326)
);
`);
```

Deze tabel heeft volgende kolommen:

- id
- refgem: unieke code voor de gemeente
- naam: de naam van de gemeente
- viewCount: aantal views per dag, standaard 0
- total_views: totaal aantal views, standaard 0
- route_count: aantal routes die door deze gemeente passeren, standaard 0
- geom: geometrie van de polygonen/gemeentegrenzen (in WGS84)
- bbox_min: coördinaten van de linker onderhoek van de “box” die de gemeente volledig omvat (in WGS84)
- bbox_max: coördinaten van de rechter bovenhoek van de “box” die de gemeente volledig omvat (in WGS84)

Vervolgens wordt elke gemeente uit “municipalities_wgs84.json” om de beurt ingelezen, en worden de refgem, naam, geometrie en boundary box coördinaten aangevuld in de databank.

```
// Construct min and max points representing the bounding box
const bboxMin = `ST_SetSRID(ST_MakePoint(${bboxCoordinates[0]}, ${bboxCoordinates[1]}), 4326)`;
const bboxMax = `ST_SetSRID(ST_MakePoint(${bboxCoordinates[2]}, ${bboxCoordinates[3]}), 4326)`;

// Insert the feature into the database
await client.query(
  `INSERT INTO gemeente (refgem, naam, geom, bbox_min, bbox_max) VALUES ($1, $2, ST_SetSRID(ST_GeomFromGeoJSON($3), 4326), ${bboxMin}, ${bboxMax})`,
  [id, naam, geometry]
);
```

De geometrie moet vooraleer deze in de databank kan worden gestoken eerst nog wat bewerkt worden: “ST_SetSRID(ST_GeomFromGeoJSON(\$3), 4326)”

- ST_GeomFromGeoJSON(\$3): Zet de geometrie in geojson formaat om naar het formaat dat PostGIS gebruikt.
- ST_SetSRID(ST_GeomFromGeoJSON(\$3), 4326): Stelt de Spatial Reference Identifier/CRS in op 4326, zijnde WGS84.

De boundary box wordt ingegeven aan de hand van 2 punten:

- bbox_min: de linkeronderhoek van de rechthoek die de gehele gemeente omvat.
- bbox_max: de rechterbovenhoek van de rechthoek die de gehele gemeente omvat.

In de database ziet dit er zo uit:

1 **SELECT** id, refgem, naam, bbox_min, bbox_max, geom **FROM** gemeente;

Data Output Messages Notifications

	id [PK] integer	refgem text	naam text	bbox_min geometry	bbox_max geometry	geom geometry
1	1	Refgem.45	Izegem	0101000020E610000...	0101000020E61000...	0106000020E610000001000000
2	2	Refgem.10	Wellen	0101000020E610000...	0101000020E61000...	0106000020E610000001000000
3	3	Refgem.1	Pelt	0101000020E610000...	0101000020E61000...	0106000020E610000001000000
4	4	Refgem.44	Hechtel-Eksel	0101000020E610000...	0101000020E61000...	0106000020E610000001000000
5	5	Refgem.25	Lo-Reninge	0101000020E610000...	0101000020E61000...	0106000020E610000001000000
6	6	Refgem.50	Hooglede	0101000020E610000...	0101000020E61000...	0106000020E610000001000000

Voor meer gedetailleerde info kan je terecht bij de commentaar in het script.

3. 3knooppuntenJSONtoTABLE.js

Indien je een andere naam/wachtwoord koos tijdens de installatie van PostGIS, kan je die hier invullen op lijn 8-13.

Om te kiezen hoeveel routes je in de databank wil steken, kan je “numberOfJsons” aanpassen op lijn 6.

Dit script voegt de knooppunten van de routes (gedownload met script “getRouteDumps.js”) aan de databank toe.

```
await client.query(`
  CREATE TABLE IF NOT EXISTS knooppunt (
    id SERIAL PRIMARY KEY,
    at_geometry GEOMETRY(Point, 4326),
    straat VARCHAR(255),
    puntnr INT,
    view_count DOUBLE PRECISION DEFAULT 0,
    download_count INT,
    gem VARCHAR(255) DEFAULT NULL,
    unique_code VARCHAR(255) UNIQUE,
    route_count INT DEFAULT 0,
    total_views INT DEFAULT 0
  )
`);
```

Daarvoor wordt, indien de tabel nog niet bestaat, de tabel aangemaakt.
Deze tabel heeft volgende kolommen:

- id
- at_geometry: De coördinaten van het knooppunt, in WGS84.
- straat: Indien er een straat beschikbaar is bij het knooppunt, wordt de straat toegevoegd.
- puntnr: Het knooppuntnummer, niet uniek.
- view_count: Voor elke route die hier passeert wordt het aantal views ervan, gedeeld door het aantal dagen dat de route al bestaat, hierbij opgeteld. Kortom: views per dag.
- download_count: Aantal downloads dat een knooppunt heeft, som van alle routes die door dat knooppunt passeren.
- gem: De gemeente waarin dit knooppunt ligt.
- unique_code: Unieke identifier van het knooppunt. Bestaat uit “[gemeente].[puntnr].[longitude,latitude]”.
- total_views: Aantal views dat een knooppunt heeft, som van alle routes die door dat knooppunt passeren.

Een JSON van een route bestaat uit een hele hoop info, maar er wordt vooral gekeken naar de “instructions”. Een instructie is 1 punt op de route en kan wel/niet een knooppunt zijn (afhankelijk van de “at” waarde: indien deze een waarde heeft, is het een knooppunt).

Per route wordt bijgehouden door welke gemeentes deze passeert.

```
// Execute the SQL query to find the gemeente name based on the geometry
const result = await client.query(
  `
  SELECT naam
  FROM gemeente
  WHERE ST_Contains(ST_Transform(geom, 4326), ST_SetSRID(ST_GeomFromText($1), 4326))
  `,
  [geometry]
);
```

Voordat een knooppunt wordt toegevoegd, komt een controle of het knooppunt nog niet bestaat.

```
// Function to check if an instruction already exists in the database
async function checkExistingInstruction(uniqueCode) {
  try {
    const result = await client.query(
      `
      SELECT COUNT(*) AS count
      FROM knooppunt
      WHERE unique_code = $1
      `,
      [uniqueCode]
    );
    return parseInt(result.rows[0].count) > 0;
  }
}
```

Indien het knooppunt al bestaat, worden de statistieken geüpdatet.

```
const updateQuery = `
  UPDATE knooppunt
  SET view_count = view_count + $1,
      download_count = download_count + $2,
      route_count = route_count + 1,
      total_views = total_views + $3
  WHERE unique_code = $4
`;
const updateValues = [viewCountPerDay, downloadCount, viewCount, uniqueCode];
await client.query(updateQuery, updateValues);
```

Indien het knooppunt nog niet bestaat, wordt het aangemaakt.

```
const insertQuery = `
  INSERT INTO knooppunt (at_geometry, straat, puntnr, view_count, download_count, gem, unique_code, route_count, total_views)
  VALUES (ST_SetSRID(ST_GeomFromText($1), 4326), $2, $3, $4, $5, $6, $7, 1, $8)
`;
const insertValues = [
  atGeometry,
  straat,
  puntnr,
  viewCountPerDay,
  downloadCount,
  gemeenteName,
  uniqueCode,
  viewCount
];
await client.query(insertQuery, insertValues);
```

Vervolgens worden de views en downloads van de route toegevoegd aan elke gemeente waar deze passeerde. Dit wordt tijdens de uitvoering bijgehouden in een variabele.

Na het doorlopen van de routes wordt de tabel “gemeente” geüpdatet met voorgenoemde variabele, die het aantal views per dag, het totaal aantal views en het aantal routes dat er passeren door elke gemeente bevat.

```
// Update the viewCount and routeCount columns for the current gemeente in the gemeente table
const updateQuery = `
  UPDATE gemeente
  SET viewCount = viewCount + $1,
      route_count = route_count + $2,
      total_views = total_views + $3
  WHERE naam = $4
`;
const updateValues = [viewCountPerDay, routeCount, viewCount, gemeenteName];
await client.query(updateQuery, updateValues);
```

Dit script bevat het zwaarste denkwerk van het geheel.

Het gaat dan vooral over het bijhouden van views per dag, het totaal aantal views, het aantal downloads en het aantal routes dat passeert door elk knooppunt en elke gemeente. Dit alles zonder duplicaten te verkrijgen of foute sommen te maken.

Voor een meer gedetailleerde uitleg kan je terecht bij de commentaar in het script.

4. 4DBtoGEOJSON.js

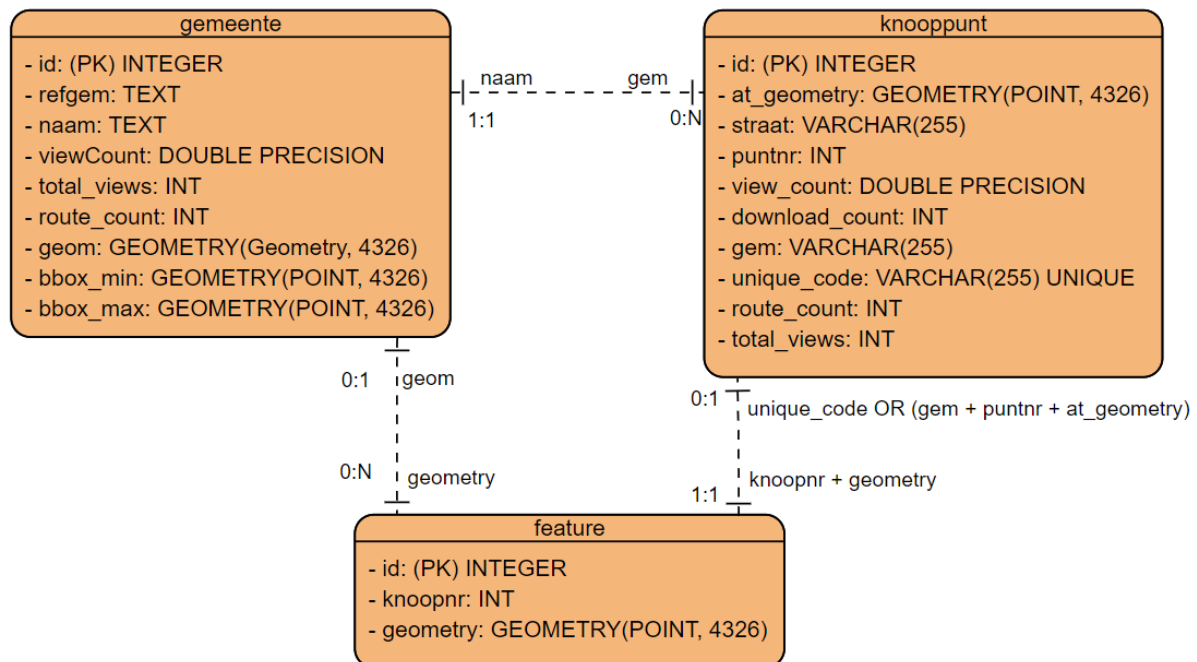
Dit is een eenvoudig scriptje dat de “gemeente” tabel uit PostGIS downloadt als GeoJSON bestand “gemFromDB.geojson”.

```
17 // Query to retrieve GeoJSON data
18 const query = `
19   SELECT
20     json_build_object(
21       'type', 'FeatureCollection',
22       'features', json_agg(ST_AsGeoJSON(t.*)::json)
23     ) AS geojson
24   FROM
25     gemeente AS t
26 `;
```

```
public
> css
1.png
2.png
3.png
4.png
5.png
circle-outline-16.png
gemFromDB.geojson
```


3. Structuur Databank

De databank bestaat hoofdzakelijk uit 2 tabellen: “gemeente” en “knooppunt”. Er is echter nog een bijkomende tabel “feature”.



gemeente

Deze tabel bevat de gegevens van alle Vlaamse gemeentes, en wordt aangemaakt in “2gemeentesJSONtoTABLE.js”. Oorspronkelijk worden enkel “refgem”, “naam”, “geom”, “bbox_min” en “bbox_max” ingevuld. De overige kolommen worden pas aangevuld na het verwerken van de routes in “3knooppuntenJSONtoTABLE.js”.

knooppunt

Deze tabel bevat alle fietsknooppunten die worden aangetroffen in de verwerkte routes. Tijdens het verwerken van deze routes worden enkele gegevens aangevuld die niet meer zullen veranderen: “at_geometry”, “straat”, “puntnr”, “gem” en “unique_code”.

Oorspronkelijk leek het aannemelijk dat de “unique_code” uniek werd door de naam van de gemeente samen te voegen met het puntnummer: “[gemeente].[puntnummer]”. Niets is echter minder waar; een gemeente kan namelijk verschillende knooppunten met hetzelfde nummer bevatten.

Om de “unique_code” dan toch uniek te maken, wordt deze nog eens aangevuld met de coördinaten van het knooppunt:

“[gemeente].[puntnr].[longitude(tot op 3 decimalen),latitude(tot op 3 decimalen)]”

Bijvoorbeeld: “Gent.3.3.723,51.002”

Er werd bewust gekozen voor 3 decimalen omdat dit ruimte laat voor kleine verschillen van enkele meters.

feature

Dit is een tabel die volledig geleegd wordt, telkens een andere gemeente/polygoon wordt aangeklikt.

Waarom wordt hier dan een Databank voor gebruikt?

Dat is omdat er op deze manier makkelijk een link kan worden gelegd met de info in de andere tabellen en om de filtratie van de features door PostGIS te laten doen.

De “feature” tabel bevat oorspronkelijk alle fietsknooppunten binnen de bbox (boundary box) van de aangeklikte gemeente, zoals die worden opgevraagd bij de API.

Daarna worden de features die zich niet binnen de grenzen van de gemeente bevinden verwijderd. Vervolgens worden ook de features die reeds in de top 5 van die gemeente staan, verwijderd.

Koppeling tabellen

gemeente – knooppunt

Een knooppunt wordt aan een gemeente gekoppeld a.d.h.v. de coördinaten van dat knooppunt, en binnen de geometrie van welke gemeente deze vallen. Hierna krijgt een knooppunt de naam van deze gemeente in de “gem” kolom.

gemeente – feature

De features worden gefilterd op basis van hun geometrie en of die wel of niet binnen een bepaalde gemeente vallen. Wanneer op een bepaald moment, wanneer er geen activiteit is in de webapplicatie, naar de databank wordt gekeken, zullen alle rijen een geometrie hebben die slechts binnen 1 gemeente valt.

knooppunt – feature

Elk knooppunt dat voorkomt in de “knooppunt” tabel heeft een equivalent in de “feature” tabel. Omdat het mogelijk is dat niet alle knooppunten in Vlaanderen werden gebruikt door de verwerkte routes, kan het dat er knooppunten in de “feature” tabel zijn die geen match hebben in de “knooppunt” tabel. Of er al dan niet een match is, wordt bepaald door de coördinaten te vergelijken tot op 3 decimalen.

Extra queries

Tijdens het ontwikkelen kwamen enkele queries van pas om bijzondere verrichtingen te doen. Deze worden niet gebruikt door de webapplicatie en zijn te vinden in de map “extra”.

save.js

Handig wanneer je per ongeluk “OprepareData.js” opnieuw bent beginnen uitvoeren en je nog niet voorbij de creatie van de “gemeente” tabel bent.

Met andere woorden: je hebt kunnen cancellen (ctrl + c) voor de verwerking van de routes begon.

Met andere woorden: de “gemeente” tabel is (deels) verwijderd/leeggemaakt.

Dit script gebruikt het bestand “gemFromDB.geojson” om de tabel opnieuw te vullen met de info van de vorige verwerking.

```

// Create table if it doesn't exist
await client.query(`
CREATE TABLE IF NOT EXISTS gemeente (
  id SERIAL PRIMARY KEY,
  refgem TEXT,
  naam TEXT,
  viewCount DOUBLE PRECISION DEFAULT 0,
  total_views INT DEFAULT 0,
  route_count INT DEFAULT 0,
  geom geometry(Geometry, 4326),
  bbox_min geometry(POINT, 4326),
  bbox_max geometry(POINT, 4326)
);
`);

console.log('Table created successfully!');

// Read the GeoJSON file
const geojsonData = JSON.parse(fs.readFileSync(geoJsonFilePath, 'utf8'));

console.log('GeoJSON Data:', geojsonData);

// Iterate through each feature and insert into the database
for (const feature of geojsonData.features) {
  const properties = feature.properties;

  const insertQuery = `
INSERT INTO gemeente (geom, refgem, naam, viewcount, total_views, route_count, bbox_min, bbox_max)
VALUES (ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), $2, $3, $4, $5, $6, ST_SetSRID(ST_MakePoint($7, $8), 4326), ST_SetSRID(ST_MakePoint($9, $10), 4326));
`;

  // Execute the query
  await client.query(insertQuery, [
    JSON.stringify(feature.geometry),
    properties.refgem,
    properties.naam,
    properties.viewcount,
    properties.total_views,
    properties.route_count,
    properties.bbox_min.coordinates[0],
    properties.bbox_min.coordinates[1],
    properties.bbox_max.coordinates[0],
    properties.bbox_max.coordinates[1]
  ]);
}

```

dropTables.js

Dit script gebruikt twee zeer eenvoudige queries om de “gemeente” en “knooppunt” tabellen te verwijderen uit de databank.

```

const { Client } = require('pg');

const pgConfig = {
  user: "ferre",
  host: "localhost",
  database: "gis",
  password: "wachtwoord",
  port: 5432, // default port for PostgreSQL
};

const client = new Client(pgConfig);

async function dropTables() {
  try {
    await client.connect();
    await client.query('DROP TABLE IF EXISTS gemeente;');
    await client.query('DROP TABLE IF EXISTS knooppunt;');
    console.log('Tables dropped successfully');
  } catch (err) {
    console.error(err);
  } finally {
    await client.end();
  }
}

dropTables();

```

4. Webapplicatie opzetten

Nu alle data is voorbereid en de databank gevuld is met de nodige gegevens, kunnen we deze data visualiseren.

De rode draad is hierbij het Node.js script “app.js”, dat alle routing van de webapplicatie verzorgt.

De **webapplicatie** wordt **beschikbaar** gemaakt door in de Terminal in Visual Studio Code de volgende command uit te voeren: “**nodemon app**”.

Om nu de **webapplicatie** te **gebruiken** surf je op hetzelfde toestel naar <http://localhost:3000/>.

app.js

Dit script verzorgt vrijwel alle verzoeken en routing die uit de browser komen:

- Luisteren naar verzoeken op de poort 3000, zijnde de poort die open staat voor verbindingen via de browser.
- Beschikbaar stellen van de bestanden in de “public” map.
- Ophalen en doorgeven van opgevraagde bestanden in de “public” map.
- Doorgeven van opdrachten voor de PostGIS DataBank.
- Foutieve URL's doorverbinden naar de 404-pagina.



```
JS app.js > ...
1  const express = require("express"); // Import express module
2  const app = express(); // Create an instance of express
3  const path = require("path"); // Import the path module
4  const getGemeenteTop5 = require("./getGemeenteTop5"); // Import getGemeenteTop5 function
5  const fetchAllKnooppunten = require("./allKnp/fetchAllKnooppunten"); // Import fetchAllKnooppunten function
6  const insertAllKnooppunten = require("./allKnp/insertAllKnooppunten"); // Import insertAllKnooppunten function
7  const getFromRadius = require("./getFromRadius"); // Import getFromRadius function
8  const fetchNetwerk = require("./fetchNetwerk"); // Import fetchNetwerk function
```

Het script maakt gebruik van een aantal imports:

- express: Verzorgt de routing, dit is het centrale zenuwstelsel van de webapplicatie, deze zorgt dat de webapplicatie online kan worden gebracht en er verzoeken kunnen worden gedaan.
- path: Zorgt dat het script toegang heeft tot de mappenstructuur.
- getGemeenteTop5: Refereert naar een ander script, dat de 5 populairste knooppunten van een gemeente teruggeeft.
- fetchAllKnooppunten: Refereert naar een ander script, dat alle fietknooppunten binnen een bepaalde boundary-box downloadt.

- insertAllKnooppunten: Refereert naar een ander script, dat de bovenstaande knooppunten filtert en teruggeeft.
- getFromRadius: Refereert naar een ander script, dat alle knooppunten binnen een meegegeven radius teruggeeft.
- fetchNetwerk: Refereert naar een ander script, dat het fietsknooppuntennetwerk downloadt.

Dit script bevat standaard functionaliteiten maar ook enkele zaken die specifiek zijn voor dit project, namelijk de functies die worden opgeroepen door de webpagina en/of de gebruiker.

Zo kan de gebruiker/browser de bovengenoemde scripts oproepen om zaken uit te voeren en eventueel ook een antwoord te krijgen.

In het commentaar bij de code staat gedetailleerde uitleg over wat er precies gebeurt.

Scripts gebruikt door app.js

1. /fetchNetwerk.js

Een callback functie die het script “fetchNetwerk.js” uitvoert.

DOEL: een json-bestand downloaden met daarin het volledige fietsknooppuntennetwerk van Vlaanderen.

Dit script maakt de eenvoudigste API call van het hele project.

Het gaat hier om de WFS server van geodata.toerismevlaanderen.be, die een hoop info over verschillende knooppuntennetwerken beschikbaar stelt.

Er wordt een statische URL geraadpleegd waarbij een geojson-bestand wordt opgeslagen als “routes.json”. Dit bestand bevat de geometrie van het volledige fietsknooppuntennetwerk in Vlaanderen en enkele omstreken.

Waarom wordt dit json-bestand niet standaard meegegeven in de “public” folder? Dat wordt zo voorzien, maar hoeft niet. Indien het bestand niet wordt gevonden wordt dit script opgeroepen om het te downloaden. Zo kan het weergegeven netwerk toch ietwat actueel worden gehouden.

```
27  module.exports = fetchRoutes;
```

Op lijn 27 staat de code die ervoor zorgt dat dit script kan worden uitgevoerd door “app.js”.

2. /getGemeenteTop5

Een callback-functie die het script “getGemeenteTop5.js” uitvoert.

DOEL: een array van de 5 knooppunten met het hoogste aantal views per dag voor een bepaalde gemeente teruggeven.

Ook dit script is vrij eenvoudig.

Er wordt een query uitgevoerd die de 5 knooppunten met de meeste views per dag van een bepaalde meegegeven gemeente teruggeeft. In de query worden de coördinaten omgezet naar het WGS84 formaat, zodat deze leesbaar en bruikbaar kunnen worden weergegeven.

```
// Query to fetch the top 5 entries with the most views from knooppunt table
const query = `
SELECT
  ST_X(ST_Transform(at_geometry, 4326)) AS lng,
  ST_Y(ST_Transform(at_geometry, 4326)) AS lat,
  *
FROM
  knooppunt
WHERE
  gem = $1
ORDER BY
  view_count DESC
LIMIT 5
`;

// Execute the query and pass along the "gemeente" name as a parameter
const result = await client.query(query, [naam]);
```

Op lijn 49 staat de code die ervoor zorgt dat dit script kan worden uitgevoerd door “app.js”.

3. /getGemNotTop5:

DOEL: teruggeven van een array van de fietsknooppunten die binnen de gemeente vallen en niet voorkomen in de top 5 van die gemeente.

Dit is een callback function die kan worden aangesproken in “app.js”.

Bij het aanspreken ervan worden enkele argumenten meegegeven:

- De naam van de gemeente
- De top 5 knooppunten van de gemeente
- De boundary box van de gemeente

De inhoud van deze callback function wordt omvat door een async Promise, om te verzekeren dat de inhoud steeds in de juiste volgorde wordt uitgevoerd.

Deze callback functie voert 2 scripts uit:

fetchAllKnooppunten.js

Eerst wordt fetchAllKnooppunten.js uitgevoerd, deze bevindt zich in de map “allKnp”.

DOEL: alle fietsknooppunten binnen de boundary box van een bepaalde gemeente downloaden.

Dit script krijgt bij activatie de boundary box van de gepaste gemeente mee.

Deze boundary box wordt meegegeven met een call naar de API/WFS van geodata.toerismevlaanderen.be.

Hiervoor moeten de coördinaten eerst worden getransformeerd van WGS84 naar Lambert72, omdat deze laatst vernoemde gehanteerd wordt door de API.

Het resultaat wordt ook in Lambert72 opgevraagd, aangezien andere coördinatensystemen niet worden ondersteund.

Na afloop wordt het opgevraagde json-bestand opgeslagen als “knoop.json” in de map “data”.

insertAllKnooppunten.js

Hier worden de knooppunten in “knoop.json” in de databank gestoken om het filteren te vergemakkelijken.

DOEL: De knooppunten filteren zodat enkel die binnen de gemeente en zij die niet voorkomen in de top 5 van die gemeente worden behouden. De resulterende rijen uit de tabel worden dan als array teruggegeven.

1: insertFeatures

Daarvoor wordt eerst de functie “insertFeatures” opgeroepen, met als argumenten:

- features: alle knooppunten die in “knoop.json” worden omvat
- client: een nieuwe client om te communiceren met de PostGIS databank

Telkens deze functie wordt opgeroepen, verwijdert deze (indien reeds aanwezig) de bestaande “feature” tabel. Daarna maakt hij deze opnieuw aan met volgende kolommen:

- id
- knoopnr: nummer van het knooppunt
- geometry: puntcoördinaten van het knooppunt, in WGS84

Vervolgens worden alle meegegeven features/knooppunten stuk voor stuk in de tabel ingevoerd.

2: filterFeaturesByGeometry

Daarna wordt “filterFeaturesByGeometry” uitgevoerd, met als argumenten:

- naam : de naam van de gemeente in kwestie
- client: een nieuwe client om te communiceren met de PostGIS databank

Deze functie verwijdert alle rijen uit de tabel “feature” die een puntgeometrie hebben die buiten de geometrie van de genoemde gemeente vallen.

```
// Delete features that do not meet the filter requirement
const result = await client.query(
  `
  DELETE FROM feature
  WHERE id NOT IN (
    SELECT f.id
    FROM feature f
    INNER JOIN gemeente g ON ST_Contains(g.geom, f.geometry)
    WHERE g.naam = $1
  )
  `,
  [gemeenteName]
);
```

3: filterFeaturesByTop5

Daarna wordt “filterFeaturesByGeometry” uitgevoerd, met als argumenten:

- top5Array: een array met de unieke code van de top 5 van de gemeente in kwestie

```
Filtering features for top 5 [
  'Aalter.75.3.449,51.150',
  'Aalter.83.3.491,51.095',
  'Aalter.74.3.453,51.127',
  'Aalter.89.3.450,51.116',
  'Aalter.88.3.469,51.111'
]
```

- client: een nieuwe client om te communiceren met de PostGIS databank

Hierbij worden de coördinaten en de knoopnummers in de unieke codes gebruikt om deze top 5 uit de “feature” tabel te verwijderen.

Ter info: een unieke code zit als volgt in elkaar:

“[gemeente].[knooppuntNummer].[longitude(3 decimalen)],[latitude(3 decimalen)]”

De toevoeging van de coördinaten is nodig aangezien een gemeente meerdere knooppunten met hetzelfde nummer kan bevatten.

```
// Delete features that meet the filter requirement
try {
  const result = await client.query(
    `
    DELETE FROM feature
    WHERE ROUND(ST_X(geometry)::numeric, 3) = $1 AND ROUND(ST_Y(geometry)::numeric, 3) = $2 AND knoopnr = $3
    `,
    [parseFloat(longitude), parseFloat(latitude), parseInt(puntNr)]
  );
}
```

4: getKnpFromDB

Daarna wordt “getKnpFromDB” uitgevoerd, met een nieuwe PostGIS client als argument.

Deze functie voert een query uit op de “feature” tabel. Deze query geeft alle resterende knooppunten uit de “feature” tabel terug, na er (indien beschikbaar) de bijhorende views per dag uit de “knooppunt” tabel aan toe te voegen. Dit gebeurt met een LEFT JOIN op basis van de coördinaten, aangezien de features enkel een knoopnummer hebben ter identificatie.

```
// Query to select all content from the features table
const query = `SELECT f.knoopnr,
  ST_X(f.geometry) AS lon,
  ST_Y(f.geometry) AS lat,
  k.view_count
  FROM feature f
  LEFT JOIN knooppunt k ON
    ROUND(ST_X(f.geometry)::numeric, 3) = ROUND(ST_X(k.at_geometry)::numeric, 3)
    AND ROUND(ST_Y(f.geometry)::numeric, 3) = ROUND(ST_Y(k.at_geometry)::numeric, 3);
`;
```

```
Retrieved content of the features table: [
  {
    knoopnr: 92,
    lon: 3.347068290975036,
    lat: 51.105356286656956,
    view_count: 3.5650705028264356
  },
]
```

Op lijn 292 staat de code die ervoor zorgt dat dit script kan worden uitgevoerd door “app.js”.

4. /getFromRadius

Een callback-functie die het script “getFromRadius.js” uitvoert.

Doel: een array van de knooppunten binnen een bepaalde straal rond een bepaald punt teruggeven.

Dit script krijgt volgende argumenten mee:

- Latitude
- Longitude
- Radius: straal in kilometer

Dit script bestaat uit 1 functie die de unieke code, views per dag, longitude en latitude van alle knooppunten binnen de meegegeven straal rond de meegegeven coördinaten teruggeeft.

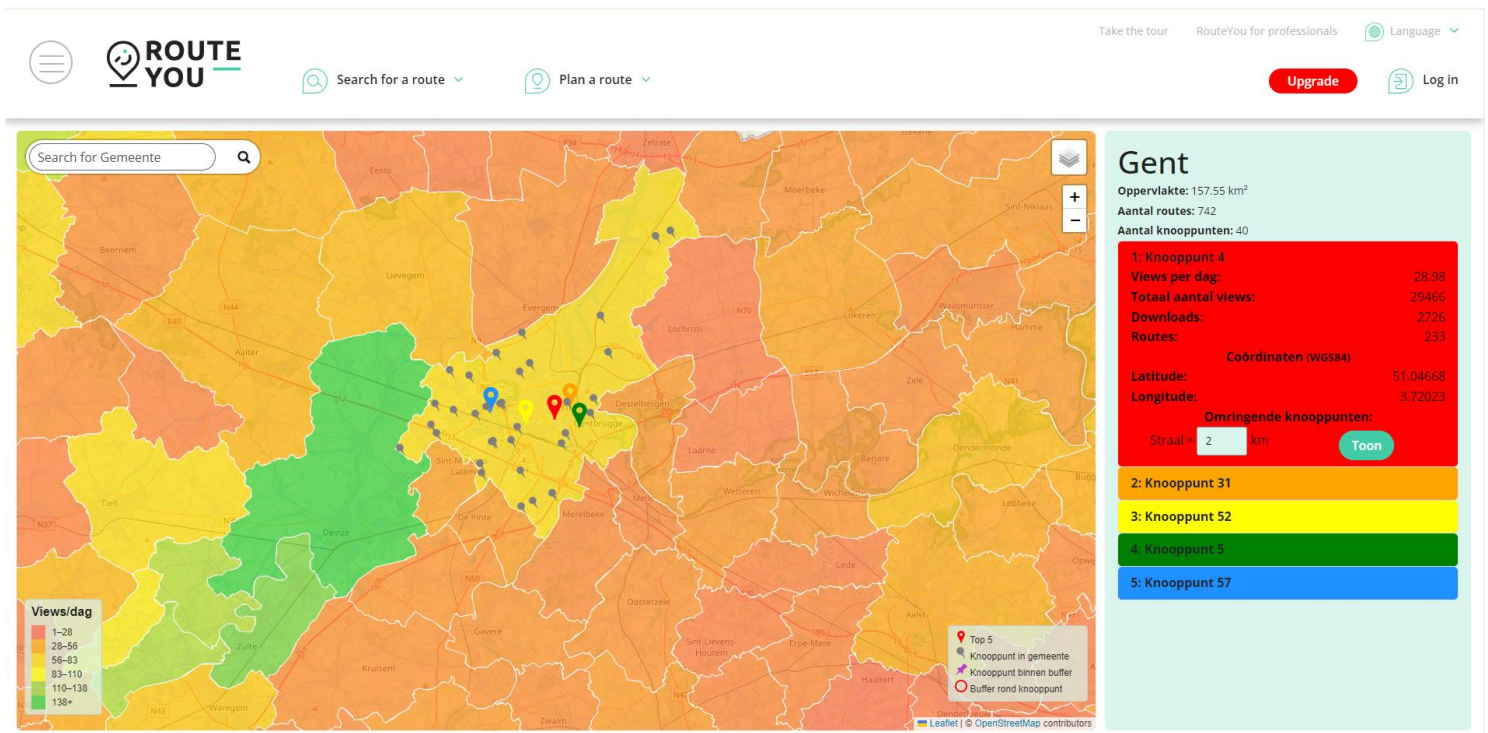
```
// Query to fetch the top 5 entries with the most views from knooppunt table
const query = `
  WITH poi AS (
    SELECT ST_SetSRID(ST_MakePoint($1, $2), 4326) AS point_of_interest
  )
  SELECT unique_code,
    view_count,
    ST_X(ST_Transform(at_geometry, 4326)) AS lon,
    ST_Y(ST_Transform(at_geometry, 4326)) AS lat
  FROM knooppunt
  CROSS JOIN poi
  WHERE ST_Intersects(
    at_geometry,
    ST_Buffer(poi.point_of_interest::geography, $3)::geometry
  )
  AND NOT ST_Equals(at_geometry, poi.point_of_interest);
`;

// Execute the query
const result = await client.query(query, [Lon, Lat, radiusInMeters]);
```

```
Knooppunten binnen: 2 km:
[
  {
    unique_code: 'Aalter.82.3.442,51.143',
    view_count: 7.330037270995067,
    lon: 3.44151,
    lat: 51.14254
  },
```

Op lijn 55 staat de code die ervoor zorgt dat dit script kan worden uitgevoerd door “app.js”.

Opmerking: het is mogelijk dat er enkele wandelknooppunten worden weergegeven bij het tonen van de knooppunten binnen de radius (deze hebben dan geen overeenkomstige grijze marker). Dit komt door gebruikers die routes uploaden als fietsroute, maar wel wandelknooppunten gebruiken.



1. html

De relevante zuivere html bevindt zich tussen lijn 2167 en 2425 van “index.html” andere html behoort tot de template die ik van RouteYou kreeg.

De eerste <div> bevat alle inhoud die door mij werd toegevoegd.

Deze <div> bevat 2 <div>’s, de eerste staat links en bevat de Leaflet kaart en alles dat daar bij hoort. De tweede staat rechts en bevat de textuele details.

De html op zich zit vrij eenvoudig in elkaar: per aangeklikte gemeente wordt het volgende weergegeven:

- Naam van de gemeente
- Oppervlakte van de gemeente, berekend a.d.h.v. de geometrie van die gemeente
- Het aantal verschillende routes dat door die gemeente loopt
- Het aantal knooppunten in die gemeente
- Een lijst met de top 5 van populairste knooppunten in die gemeente
 - o Per item in de lijst wordt standaard het rangnummer en knooppuntnummer
 - o Wanneer op een item in de lijst wordt geklikt, wordt de ingesloten tabel zichtbaar.

Deze bevat:

- Indien beschikbaar: de straatnaam (meestal niet beschikbaar)
- Views per dag
- Totaal aantal views
- Totaal aantal downloads
- Aantal routes dat langs dit knooppunt passeert
- Coördinaten (klik om te kopiëren)
- Een invoerveld voor een straal en een knop om een buffer met deze straal en de omringende knooppunten rond het geselecteerde knooppunt te tonen.

Op het linker paneel vind je de Leaflet-kaart, met daarop nog enkele overlays:

- Linksboven: zoekbalk voor gemeentes
- Rechtsboven:
 - o Controlepaneel voor weergaven van kaartlagen
 - o Zoom-knoppen
- Linksonder: legende voor de kleuren op de choropletenkaart van de gemeente
- Rechtsonder: legende voor de symbolen op de kaart

Onder de divs staat, op vraag van de docenten ter informatie nog eens hoeveel routes er werden gebruikt om deze kaart te creëren, en tot hoe ver terug deze dateren.

2. JavaScript

Lijn 2437 – 2540: Initialisatie Leaflet kaart

- Aanmaken van de Leaflet kaart, OpenStreetMap toevoegen en de view instellen.
- De gebruikte lagen aanmaken en toevoegen (behalve de routesLayer, die wordt standaard niet weergegeven).
- Layer control toevoegen om te kunnen kiezen welke lagen worden getoond.
- Search control toevoegen: hiermee zoek je een gemeente op aan de hand van de naam in plaats van die te zoeken op de kaart.
- Legende voor gebruikte symbolen toevoegen.

Lijn 2543 - 2839: Aanmaak en weergave van Gemeentes als polygonen

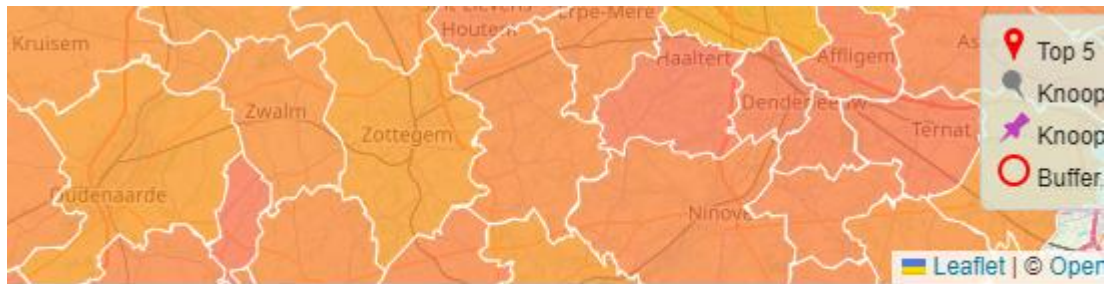
- Ophalen van geojson van gemeentes.
- Gemeentes een kleur geven, bepaald a.d.h.v.. aantal views/dag.
- Elke gemeente/polygoon de nodige event listeners geven voor: tooltips, weergave van details in detailpaneel, weergaven van de top 5 van die gemeente in het detailpaneel.
- Uiteindelijk wordt ook de weergave van de knooppunten die niet in de top 5 horen getriggerd.
- Per knooppunt in de top 5 (bij kleine datasets is het mogelijk dat dit er minder dan 5 zijn) de detailtabel aanvullen met info en een event listener instellen om de tabel wel/niet te tonen wanneer het item wordt aangeklikt.
- Aanmaken van de legende voor de choropletenkaart (kleur plogygonen/gemeentes).

Lijn 2842 – 2895: Knooppuntennetwerk toevoegen

- Json met fietsroutes tussen de knooppunten ophalen van server.
- Indien niet beschikbaar worden deze gedownload van de API van geodata.toerismevlaanderen.be
- Daarna worden de fietsroutes toegevoegd aan de kaart, maar zoals reeds gezegd, niet standaard weergegeven.

Lijn 2897 – 2904: Extra info weergeven

- Ophalen van een klein .txt bestand van de server.
- Weergave van het aantal routes die verwerkt werden voor de weergave van de huidige kaart en hoe oud deze routes kunnen zijn.



15363 routes, vanaf 2020-01-01

Lijn 2907 – 2955: Markers voor Top 5 toevoegen

- Getriggerd door het aanklikken van een gemeente/polygoon.
- Weergave van de top 5 populairste fietsknooppunten in verschillende kleuren op de kaart.
- Event listener voor hoveren toevoegen.
- Event listener voor klikken: uitvouwen van tabel in lijst in detailpaneel (door de klik om te leiden naar een klik op zo'n item in de lijst).

Lijn 2957 – 2996: Markers toevoegen voor knooppunten die NIET in Top 5 zitten

- Weergeven van totaal aantal knooppunten in gemeente.
- Voor elk knooppunt een marker maken.
- Info toevoegen aan tooltip van die marker.

Lijn 2998 – 3069: Markers binnen Buffer

- Ophalen van knooppunten (over gemeentegrenzen heen) binnen de gewenste bufferafstand/straal rond een bepaald Knooppunt uit de top 5.
- Aanmaken van markers met tooltips met details.
- Weergave van deze markers en de radius.
- Wissen van deze laag wanneer iets anders wordt aangeklikt.

Lijn 3072 – 3089: Coördinaten kopiëren

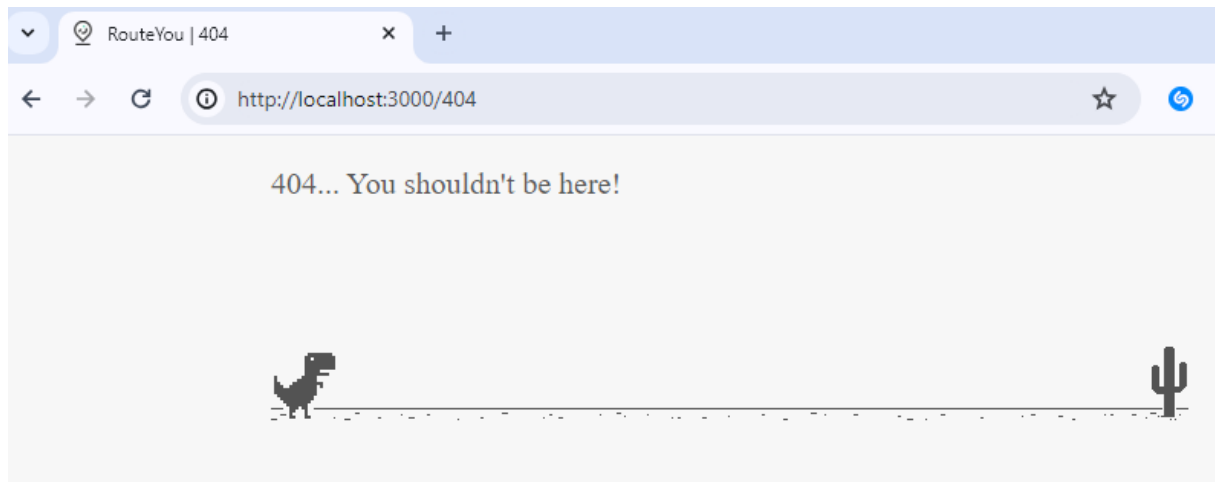
- De coördinaten van de Knooppunten in de top 5 kunnen worden gekopieerd door erop te klikken.

Lijn 3092 – 3094: Klik stoppen

- Wordt gebruikt door de inputvelden van de Straal van de buffer, zodat klikken op zo'n inputveld niet resulteert in de tabel die ingevouwen wordt.

404.html

Wanneer je naar een onbestaande URL op de site probeert te surfen, wordt je omgeleid naar de 404-pagina.



Ik ben op geen enkele manier de auteur of eigenaar van het T-Rex spel.

Bron van deze code: <https://codepen.io/MysticReborn/embed/rygqao?>

Copyright (c) 2014 The Chromium Authors. All rights reserved.

Veelgebruikte bronnen

- RouteYou Webservice: voor routedump bestanden.
- PostGIS documentatie: voor samenstellen van queries.
 - o <https://postgis.net/docs/>
- Leaflet documentatie: <https://leafletjs.com/reference.html>
- Iconen/symbolen op de kaart: www.iconsdb.com
- [StackOverflow](#): voor specifieke problemen.
- Turf.js: om oppervakte van polygonen te bepalen.
 - o <https://turfjs.org/getting-started/>
- [transform-coördinates](#)
- Leaflet search: zoekfunctie voor gemeentes op de kaart
 - o <https://opengeo.tech/maps/leaflet-search/>
- Node.js Crash Course van Net Ninja op YouTube:
 - o Routing en app.js
 - o Opzetten van een Node.js omgeving
 - o <https://youtube.com/playlist?list=PL4cUxeGkcC9jsz4LDYc6kv3ymONOKxwBU&si=eDHtRLSSiDGKQPBX>
- HTML en CSS van RouteYou.com
- AI: ChatGPT en CoPilot
- API's:
 - o geodata.toerismevlaanderen.be voor fietsknooppunten en fietsknooppuntennetwerk
 - o api.routeyou.com (niet openbaar) voor routes
 - o geo.api.vlaanderen.be voor referentiebestand Vlaamse gemeenten