**Stress-Map Documentation**

Pre-process extract

Pre-process the extract and start a routing engine HTTP server on port 5000.

Depending on what kind of routing you want to use, choose the profile accordingly:

car: car.lua

walking: foot.lua

bike: bicycle.lua

***docker run -t -v c:/docker:/data osrm/osrm-backend osrm-extract -p /opt/PROFILE.lua /data/CITY-latest.osm.pbf***

So for car rides in Berlin use:***docker run -t -v c:/docker:/data osrm/osrm-backend osrm-extract -p /opt/car.lua /data/berlin-latest.osm.pbf***

Then run:***docker run -t -v c:/docker:/data osrm/osrm-backend osrm-partition /data/berlin-latest.osrm***

***docker run -t -v c:/docker:/data osrm/osrm-backend osrm-customize /data/berlin-latest.osrm***

**Why we setup Docker Image**

Because extracting, customizing osm.pbf that is map of any particular region is not suitable for desktop machine requires lot of computational power and RAM so Docker

**How to Setup OSRM Docker Image**

1. Refer this article

<https://gist.github.com/AlexandraKapp/e0eee2beacc93e765113aff43ec77789>

* 1. Start by installing docker on your desktop
  2. Pulling the image
  3. Download OpenStreetMap extract (:I downaloaeded onlot Atlanta, I suggest to downalod all of United States of America - <https://download.geofabrik.de/north-america/us.html>
  4. Start Pre-Processing since it is important for to create paths for Bicycles, Foot and Car
  5. Start Routing Engine  
       
     docker run --name osrm -t -i -p 5000:5000 -v c:/docker:/data osrm/osrm-backend osrm-routed --algorithm mld /data/berlin-latest.osrm

* 1. To test request a based on profile use processed change driving to desired

curl "<http://127.0.0.1:5000/route/v1/driving/13.388860,52.517037;13.385983,52.496891?steps=true>"

* 1. Refer Installation of OSRM Server document pdf if faced with any issues
  2. Docker start “nameofyourdocker” Docker stop “nameofyourdocker”

1. Other alternatives Valhalla / Mapbox / Graphhopper
   1. Documentation

**OSRM Open Source Routing Machine API**

OSRM Response

****

{

"code": "Ok", // Indicates the status of the response (successful)

"matchings": [ // An array of matched routes

{

"confidence": 0.922436, // Confidence level of the match

"geometry": { // Route geometry

"coordinates": [ // Array of coordinates defining the route

[-84.365211, 33.777663], // Longitude and latitude of the first coordinate

[-84.364595, 33.777672] // Longitude and latitude of the second coordinate

],

"type": "LineString" // Type of geometry (LineString)

},

"legs": [ // Information about route legs

{

"annotation": { // Annotation for this leg

"metadata": { // Metadata about the annotation

"datasource\_names": [ // Names of data sources used

"lua profile"

]

},

"nodes": [69244349, 1940792548], // Nodes along the route

"datasources": [0], // Data sources used

"speed": [4.2], // Speed of the route (in meters per second)

"weight": [13.6], // Weight of the route

"duration": [13.6], // Duration of the route (in seconds)

"distance": [56.958883] // Distance of the route (in meters)

},

"steps": [], // Steps along this leg (empty in this case)

"distance": 57, // Total distance of the leg (in meters)

"duration": 13.6, // Total duration of the leg (in seconds)

"summary": "", // Summary of the leg (empty in this case)

"weight": 13.6 // Weight of the leg

}

],

"distance": 57, // Total distance of the matched route (in meters)

"duration": 13.6, // Total duration of the matched route (in seconds)

"weight\_name": "duration", // Name of the weight (duration in this case)

"weight": 13.6 // Weight of the matched route

}

],

"tracepoints": [ // Tracepoints representing the matched coordinates

{

"alternatives\_count": 85, // Number of alternative matches

"waypoint\_index": 0, // Index of the waypoint

"matchings\_index": 0, // Index of the matching

"location": [-84.365211, 33.777663], // Longitude and latitude of the tracepoint

"name": "Drewry Street Northeast", // Name of the street at the tracepoint

"distance": 13.754717, // Distance from the original coordinates to this tracepoint (in meters)

"hint": "218zgO5fMwBIAAAAkAAAAAAAAAAnAAAAaVjyQSQhcEIAAAAA\_UuDQUgAAACQAAAAAAAAACcAAAAUAAAAZbD4-v9nAwJjsPj6e2gDAgAAnwXlPuR3" // Hint for snapping to a street segment

},

{

"alternatives\_count": 207,

"waypoint\_index": 1,

"matchings\_index": 0,

"location": [-84.364595, 33.777672],

"name": "Drewry Street Northeast",

"distance": 0.776406,

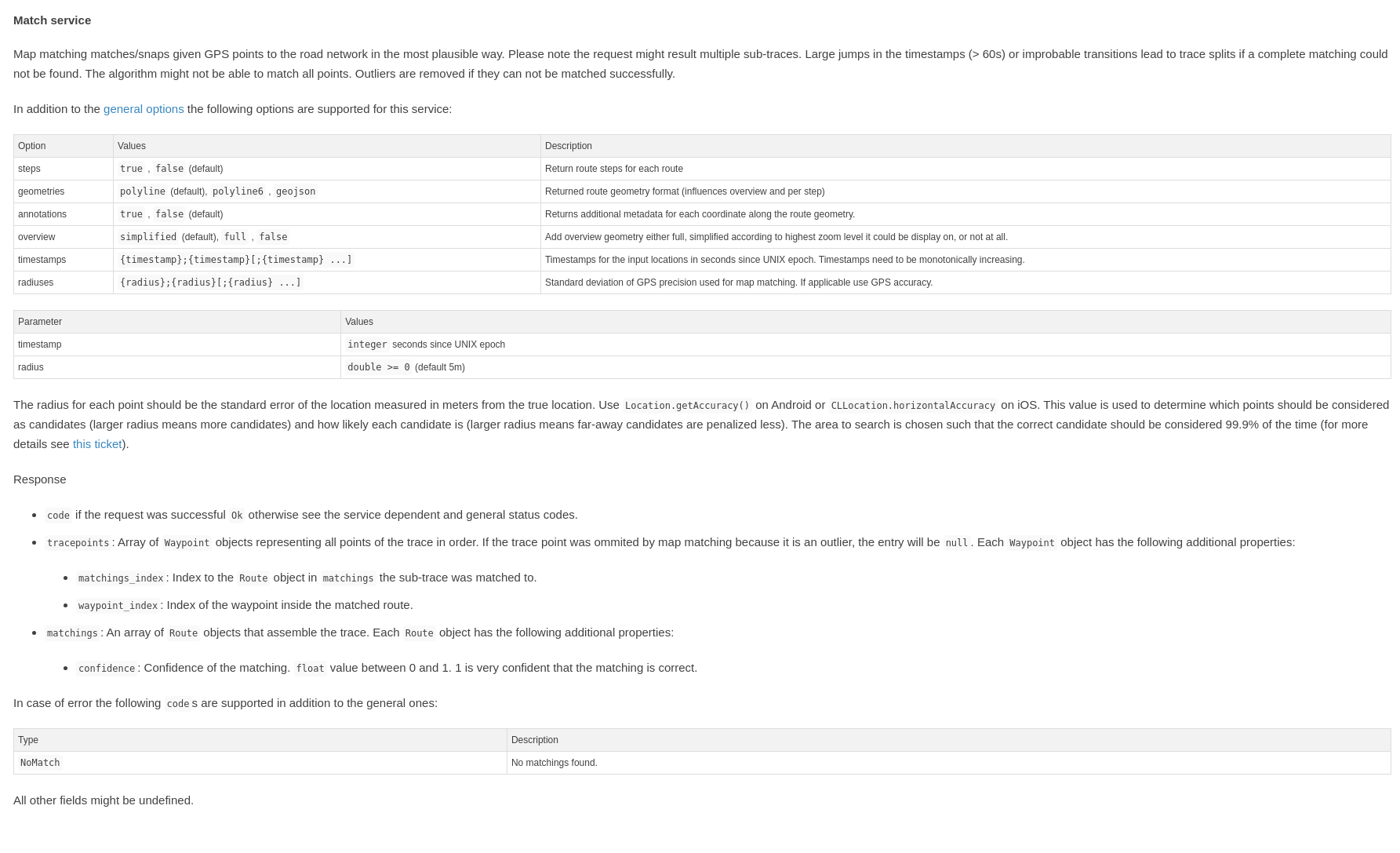
"hint": "218zgO5fMwDQAAAACAAAAAAAAAAnAAAANbmuQqi0PUAAAAAA\_UuDQdAAAAAIAAAAAAAAACcAAAAUAAAAzbL4-ghoAwLNsvj6D2gDAgAAnwXlPuR3"

}

]

}

1. OSRM documentation can be found on this link - <https://project-osrm.org/docs/v5.5.1/api/#general-options>
   1. **Match Service** - <https://project-osrm.org/docs/v5.5.1/api/#match-service>



We used match service to align the group of coordiantes from SQL server to route it (snapped) most efficient way using this service. It takes maximum of 90 to 100 coordiantes in one API call.

Example

<http://localhost:5000/match/v1/bicycle/-84.365213,33.777787;-84.364595,33.777679?overview=full&radiuses=49;49&geometries=geojson&tidy=true&annotations=true>

Annotations: it consists of metadata about the coordinates revived from API that is data about the snapped coordinates to the road. It inlcudes nodes,

"nodes": [2256317495, 69331811, 8155111912],

"datasources": [0, 0],

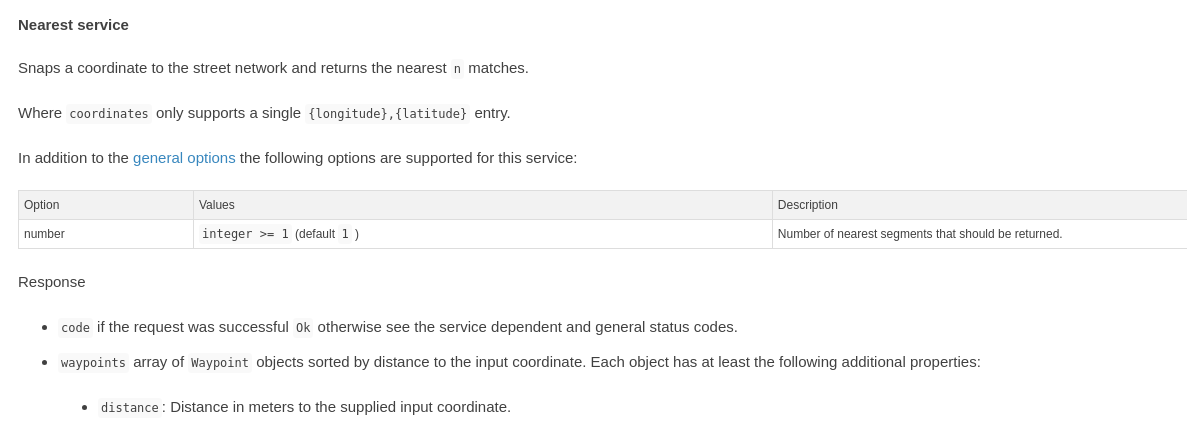
"speed": [4.2, 4.2],

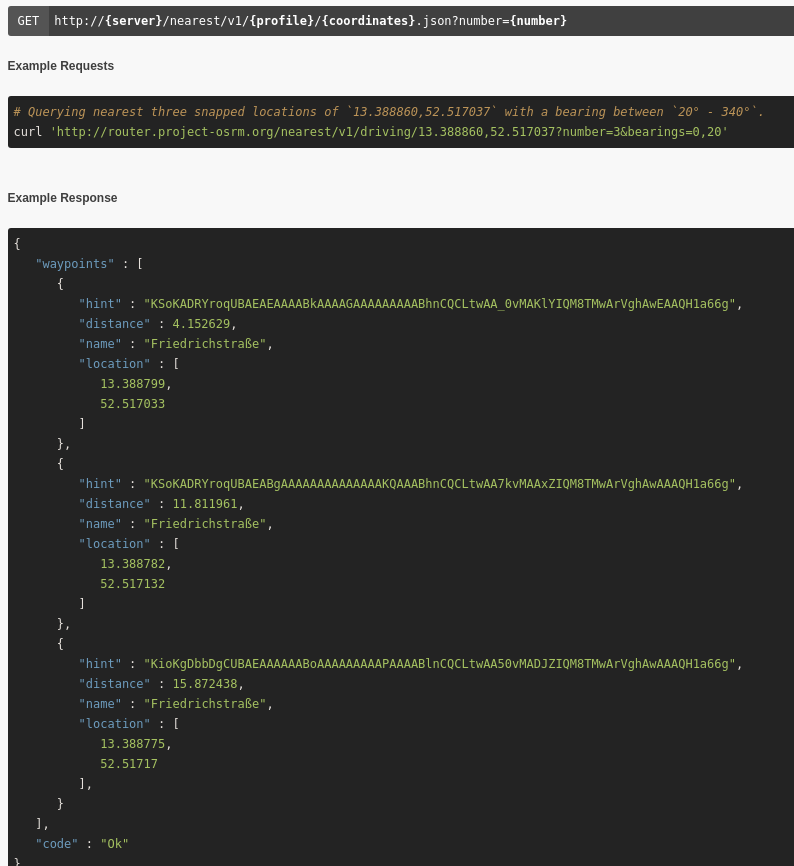
"weight": [14.5, 7.4],

"duration": [14.5, 7.4],

"distance": [60.380708, 31.030934]

Properties

* distance: The distance, in metres, between each pair of coordinates
* duration: The duration between each pair of coordinates, in seconds
* datasources: The index of the datasource for the speed between each pair of coordinates. 0 is the default profile, other values are supplied via --segment-speed-file to osrm-contract
* nodes: The OSM node ID for each coordinate along the route, excluding the first/last user-supplied coordinates
  1. Nearest Service – It could be used to find out nearest ways, roads, paths



* 1. Route Service
  2. Trip Service
  3. Tile Service
  4. Table Service

Route Service: Provides functionality to calculate routes between two or more locations, considering various factors like distance, duration, traffic conditions, etc.

Applications: Navigation and routing apps: Helps users find the shortest or fastest route between two points, considering traffic conditions and other constraints. Logistics and fleet management: Optimizes delivery routes to minimize time and fuel consumption, improving efficiency. Emergency services: Helps emergency responders find the quickest route to reach a location in case of emergencies.

Trip Service: Manages and analyzes trip data, including GPS traces, timestamps, and other relevant information.

Applications: Transportation analysis: Analyzes travel patterns and behaviors to understand traffic flow, congestion hotspots, and commuting patterns. Tourism planning: Helps tourism agencies analyze visitor movement patterns to optimize tourist routes and attractions placement. Environmental impact assessment: Studies the impact of transportation on the environment by analyzing trip data to identify areas with high emissions or congestion.

Tile Service: Provides raster map tiles for displaying maps on web or mobile applications.

Applications: Web mapping applications: Displays interactive maps on websites and mobile apps, allowing users to explore geographic data and visualize information. Location-based services: Integrates maps into applications to provide location-aware features like finding nearby restaurants, businesses, or points of interest. Geographic information systems (GIS): Supports spatial analysis and visualization by providing map layers as raster tiles, enabling users to overlay and analyze different datasets.

Table Service: Manages and stores tabular data associated with geographic features, such as attributes of map features or spatial data.

Applications: Data management: Stores and organizes geographic data in structured tables, allowing efficient retrieval and management of spatial information. Geographic data analysis: Provides a platform for analyzing and querying spatial data attributes, enabling users to derive insights and make data-driven decisions. Asset management: Stores information about assets like infrastructure, utilities, or property boundaries, facilitating asset tracking, maintenance, and planning. Land management: Supports land administration by storing information about land parcels, ownership, and land use, helping governments and organizations manage land resources effectively.

**OSM Open Street Map API**

1. Open Street -

Communities to Join - <https://www.openstreetmap.org/communities>

**Architecture**

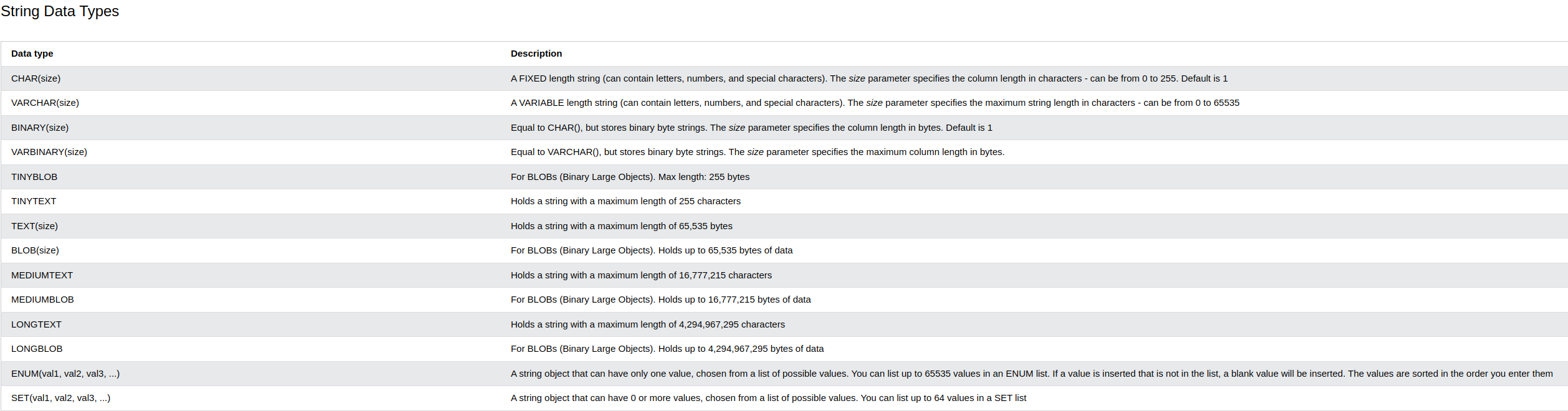
1. Polyglot
   1. Python & Py Notebooks
   2. Express, Node Javascript / Typescript
   3. SQL

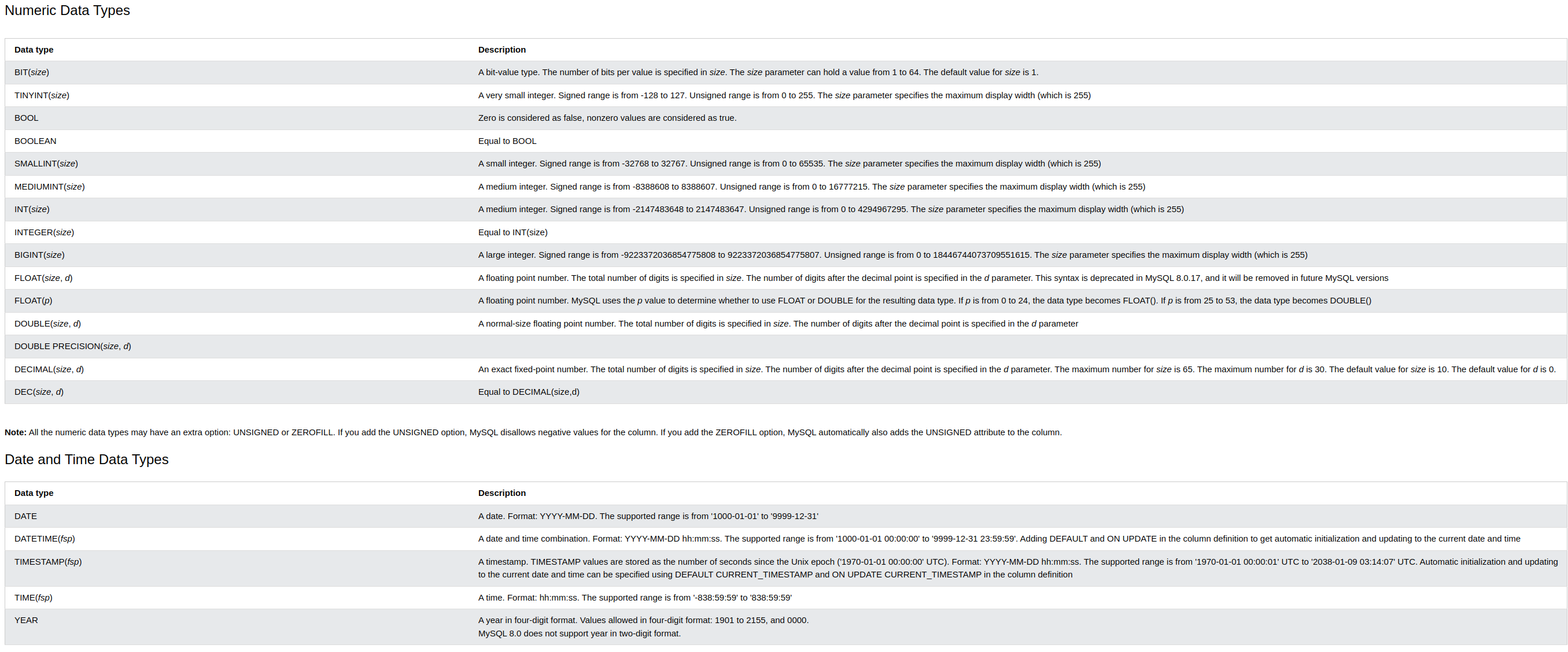
|  |  |  |  |
| --- | --- | --- | --- |
| Data Extraction and Analysis | Python & Py Notebooks (pyosrmplay2.pynb) | OSRM API for routing | OSM API for Nodes |
| MySQL Table (st – database, osr table) | | |
| Translation // Backend | Node & Express Javascript Loading MySQL table (server.js),  Previous works routing using API (server.js) | | |
| Frontend | Front end display of routes (MyMap.tsx, Snap.tsx) | | |

**SQL Table**

I am writing about datatypes here that we used to store the information from OSRM & OSM TEXT, LONGTEXT, VARCHAR, INT.

Below screenshots are about the description and size of data types for future reference.





**To start MySQL on Linux:**

**My System:** sudo mysql -u root -p (password is root)

**if not root just** press enter

USE DatabaseName;

Here, the **DatabaseName** is the name of the database that we want to select

CREATE DATABASE testDB;

SHOW DATABASES;

**to input database:** “soruce /filepath”

**To install & start XAMPP:**

How do I install XAMPP?

Choose your flavor for your linux OS, the 32-bit or 64-bit version.

Change the permissions to the installer

chmod 755 xampp-linux-\*-installer.run

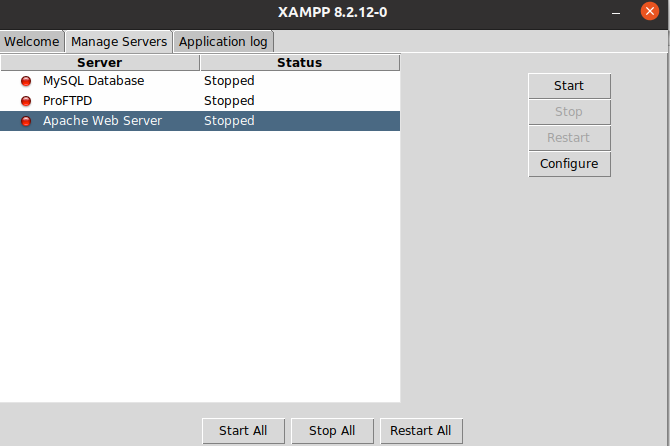
Run the installer

sudo ./xampp-linux-\*-installer.run

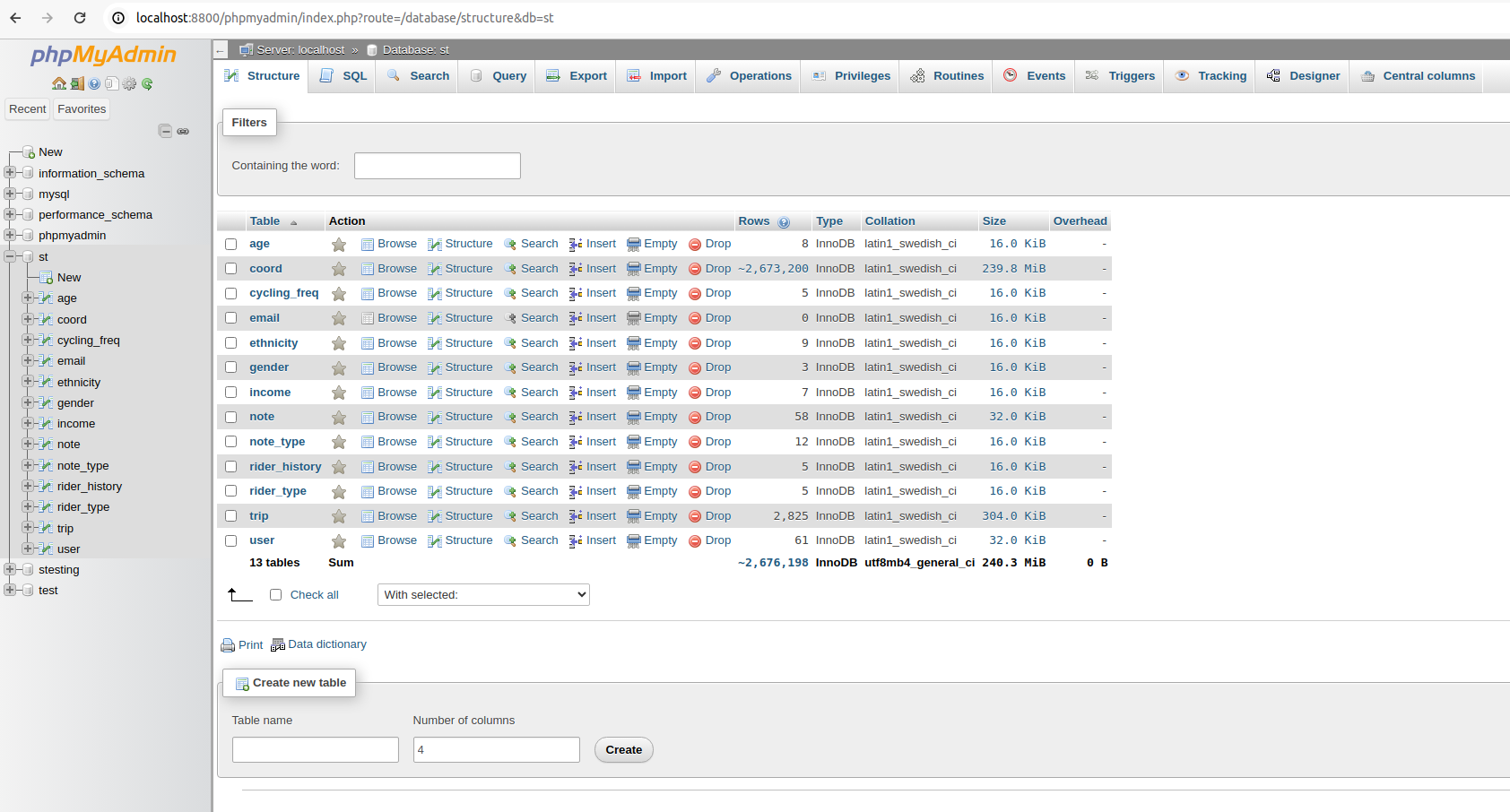
That's all. XAMPP is now installed below the /opt/lampp directory.

cd *opt*lampp/

sudo ./manager-linux-x64.run



start Apache Web Server and start MySQL Database then go to localhsot phpyadmin to access and see the SQL table



**To access XAMPP/MariaDB through terminal on Linux:**

“sudo /opt/lampp/bin/mysql -h localhost -u root -p”

**to input database:** “soruce /filepath”

**CODE**

**Snap.tsx**

Explanation:

1. This component renders a map using Leaflet library to display trip data fetched from a backend API.
2. It consists of two **useEffect** hooks:
   1. The first **useEffect** hook fetches trip data from the backend API when the **tripId** state changes.
   2. The second **useEffect** hook processes and displays trip data on the map when either the **trip** or **step** state changes.
3. The **createPairs** and **createPairsmini** functions create pairs of coordinates from the trip data based on the step size.
4. The **sendOsrmRequest** function sends a request to the OSRM API to get route information based on the coordinates.
5. The **plotPolyline** function plots the polyline on the map using Leaflet.
6. The **divideTripIntoParts** function divides the trip data into parts and processes each part.
7. The **handleStepChange** and **handleTripIdChange** functions handle changes in the step and trip ID inputs, respectively.
8. The component returns JSX elements including the map container, input range for selecting step, and select dropdown for selecting trip ID.

**MyMap.tsx**

1. **Imports**: The component imports necessary modules from React, Axios for making HTTP requests, Leaflet for rendering maps, and defines interfaces for various data types.
2. **State Variables**: The component defines several state variables using the **useState** hook to manage the component's state:
   * **trips**: A state variable to store trip data fetched from the backend API.
   * **currentTripIds**: An object storing the start and end indices of the current set of trips being displayed.
   * **selectedTripId**: A state variable to store the ID of the currently selected trip.
   * **tripinfo**: A state variable to store trip information fetched from the backend API.
   * **selectedTripInfo**: A state variable to store the information of the currently selected trip.
   * **userinfo**: A state variable to store user information fetched from the backend API.
   * **selectedUserInfo**: A state variable to store the information of the currently selected user.
   * **attributeTypes**: A state variable to store the types of various attributes extracted from user information.
3. **Effect Hooks**:
   * The component uses **useEffect** hooks to fetch trip data, trip information, and user information from the backend API when the component mounts (**[]** dependency array indicates it runs only once).
   * Another **useEffect** hook runs when **trips**, **currentTripIds**, or **selectedTripId** change. It plots trip routes on the map based on the current set of trip IDs.
   * Inside the **useEffect** hook, the **plotPolyline** function is called to plot the trip routes on the map.
4. **Helper Functions**:
   * **plotPolyline**: A function to plot trip routes on the map using Leaflet based on the current set of trip IDs.
   * **processOsrmData**: A function to process OSRM API response and append coordinates to allLatLngs.
   * **getRandomColor**: A function to generate a random color for trip routes.
5. **Event Handlers**:
   * **handleNext**: Handles clicking on the Next button to display the next set of trips.
   * **handlePrevious**: Handles clicking on the Previous button to display the previous set of trips.
   * **handleTripSelection**: Handles selecting a trip ID from the dropdown menu and updates the selected trip information.
   * **handleUserSelection**: Handles selecting a user ID from the dropdown menu and updates the selected user information.
6. **Rendering JSX**:
   * The component returns JSX elements to render the map, navigation buttons, trip and user selection dropdowns, trip and user information panels, and filters.

**pyosrmplay1.pynb**

**pyosrmplay2.pynb**

**Linux CMD Commands**

history | grep mysql – to check all the commands with mysql in it

history | grep docker - to check all the commands with docker in it

**To start MySQL:**

sudo mysql -u root -p (password is root)

**To start XAMPP:**

cd *opt*lampp/

sudo ./manager-linux-x64.run

**To access XAMPP/MariaDB through CMD:**

sudo /opt/lampp/bin/mysql -h localhost -u root -p

to input database: soruce …….filepath

**Download and Install XAMPP -** <https://www.youtube.com/watch?v=HJl2ILUfBoA>

<https://linux.how2shout.com/how-to-start-xampp-in-ubuntu-using-the-command-line/>

**Download Docker CE and Docker Desktop -** <https://www.youtube.com/watch?v=ILdziITdSag>

<https://www.youtube.com/watch?v=JsXNBIsFzu4>

**Few Common Erros Faced:**

Require Error:

*const express = require("express"); ^ ReferenceError: require is not defined in ES module scope, you can use import instead This file is being treated as an ES module because it has a '.js' file extension and '/backend/package.json' contains "type": "module". To treat it as a CommonJS script, rename it to use the '.cjs' file extension.*

<https://github.com/vercel/next.js/issues/24334>

<https://learn.coderslang.com/0021-nodejs-require-is-not-defined-error/>

**Open GraphHopper on Localhost:**

docker run -p 8989:8989 -v "$(pwd)/data:/data" israelhikingmap/graphhopper --input /data/georgia-latest.osm.pbf --host 0.0.0.0 --config /data/config-graphhopper.yml