**Muhammad Arslan**

**BE\_20 IGIS\_SCEE**

**Web GIS Lab**

**Google Maps API**

1. **In what situations Google Maps API is preferred?**

Google Maps API is preferred in situations where developers need access to high-quality maps, extensive geographic coverage, rich location-based data sets, easy integration, robust technical support, and cost-effective pricing plans. Common scenarios where Google Maps API is preferred include mapping and navigation, location-based services, real-time tracking, data visualization, and analytics and reporting. Compared to open-source mapping solutions, Google Maps API offers unique advantages such as up-to-date maps, comprehensive data sets, and easy-to-use APIs, making it a versatile tool for building location-based applications.

1. **What are other mapping API’s available in competition with Google Maps API?**

There are several mapping APIs available that are in competition with Google Maps API. Some of the most popular ones include:

Mapbox: Mapbox is a popular mapping platform that offers highly customizable maps and location-based data services. Mapbox APIs are designed to make it easy to integrate into web and mobile applications.

OpenStreetMap: OpenStreetMap is a free and open-source mapping platform that provides global coverage with detailed street maps and points of interest. OpenStreetMap relies on contributions from its user community to build and maintain its maps and data, making it a collaborative and community-driven alternative to Google Maps API.

**Leaflet** and **OpenLayers** are two additional mapping libraries that are commonly used in conjunction with open-source mapping data sources like OpenStreetMap. These libraries provide a customizable and flexible solution for building interactive maps and location-based applications. Leaflet is known for its lightweight and easy-to-use API, while OpenLayers offers a more feature-rich and powerful mapping solution. Both Leaflet and OpenLayers can be used as alternatives to Google Maps API, particularly for projects where open-source data sources and customization are preferred.

Bing Maps API: Bing Maps API is a mapping platform developed by Microsoft that provides geocoding, routing, and traffic data services.

HERE Maps API: HERE Maps API is a mapping platform that provides high-quality maps, geocoding, routing, and traffic data services.

TomTom Maps API: TomTom Maps API is a mapping platform that provides geocoding, routing, and traffic data services.

1. **Can you code Google Maps API in jQuery framework as opposed to simple JavaScript?**

Added the jquery cdn:

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

Google Maps API Key:

<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBmWvG0umKXjtmkkVN1w9bqBrNihXX3-Rk&callback=myMap"></script>

    <style>

**Jquery Code:**

    <script>

      $(document).ready(function () {

        // Initialize the map

        var map = new google.maps.Map(document.getElementById("map"), {

          center: { lat: 33.68, lng: 73.04 },

          zoom: 8,

        });

      });

    </script>

1. **Write a simple program which reads in coordinates (take any 10 of your choice) from PostgreSQL database and show them as markers on Google Maps using Google Maps API? You may need to use AJAX and PHP for this task. You should show this program during the lab. You can assume a max of ten coordinates (POI’s in Islamabad). You can do the same task in simple JavaScript or JQuery. The choice is your’s.**

Here is the map on which I showed the 10 location points fetched from POSTGIS database.

Source code:

**All the source code is Available in zip file.**

Note: Map shows “For development purposes only” which is because I am using the free version of Google Maps API Key.

**Code shortcuts:**

Connection to PostGIS Database using PHP.

<?php

$dbname = 'postgres';

$user = 'postgres';

$password = 'mas786mas';

$host = 'localhost';

$port = '5432';

// Connect to the PostGIS database

$conn = pg\_connect("dbname=$dbname user=$user password=$password host=$host port=$port");

// Check if the connection was successful

if (!$conn) {

    echo "An error occurred.\n";

    exit;

Data Fetching code:

// Fetch the location points from the database

    $result = pg\_query($conn, "SELECT ST\_X(geom) AS lng, ST\_Y(geom) AS lat FROM location\_data.points");

Storing the location points (lat, long) in an array in json format.

 // Create an array to store the location points

        $locationPoints = array();

        // Loop through the rows returned by the query

        while ($row = pg\_fetch\_assoc($result)) {

            // Add the location point to the array

            $locationPoints[] = $row;

        }

        // Return the array as JSON

        echo json\_encode($locationPoints);

    }

Adding marker on that locations Stored in the array in json format.

// Fetch the location points from the PHP script

  fetch("fetch\_data.php")

    .then((response) => response.json())

    .then((data) => {

      // Loop through the location points and add markers to the map

      data.forEach((point) => {

        var marker = new google.maps.Marker({

          position: {

            lat: parseFloat(point.lat),

            lng: parseFloat(point.lng)

          },

          map: map,

          title: point.name,

        });

The additional functionality I added is:

When I click any marker, it increases the zoom level of the location to have a better zoomed in view and I set the timeout of 5 seconds which gets the map back to its default zoom level which I set in Google map design process.

marker.addListener("click", function() {

          // Increase zoom level

          map.setZoom(12);

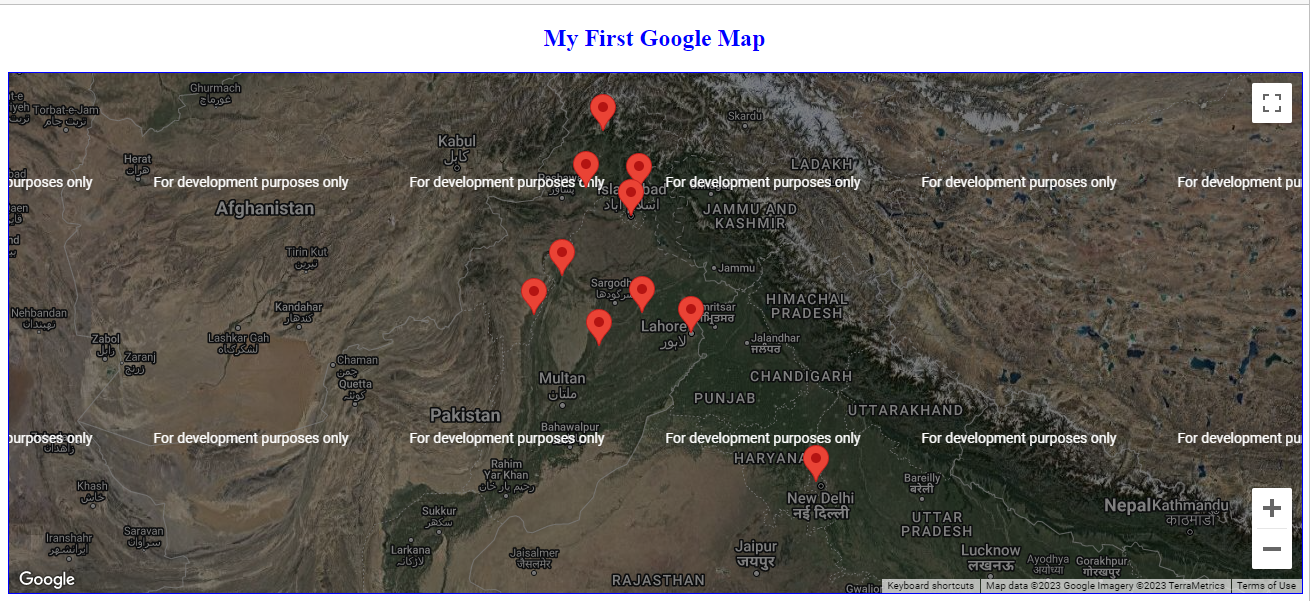
          // After 5 seconds, set zoom level back to default

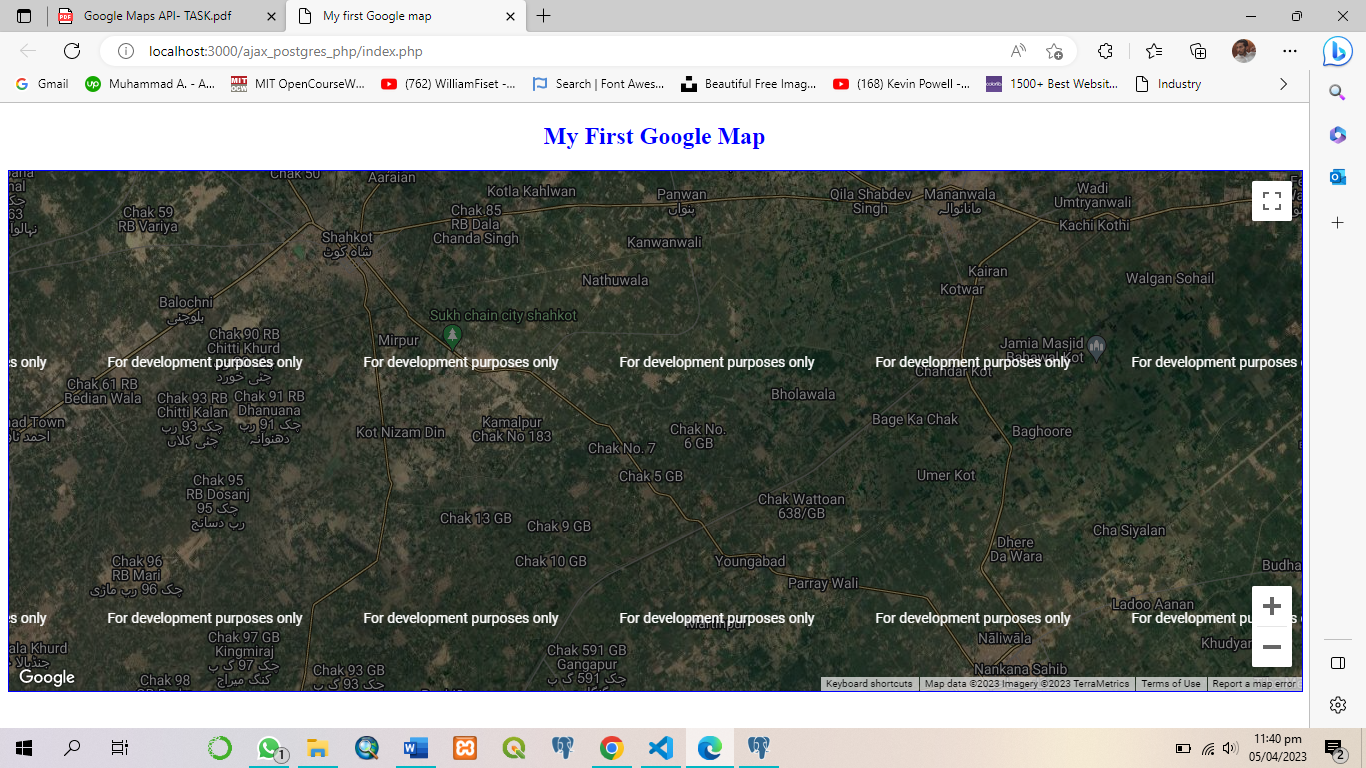
          setTimeout(function() {

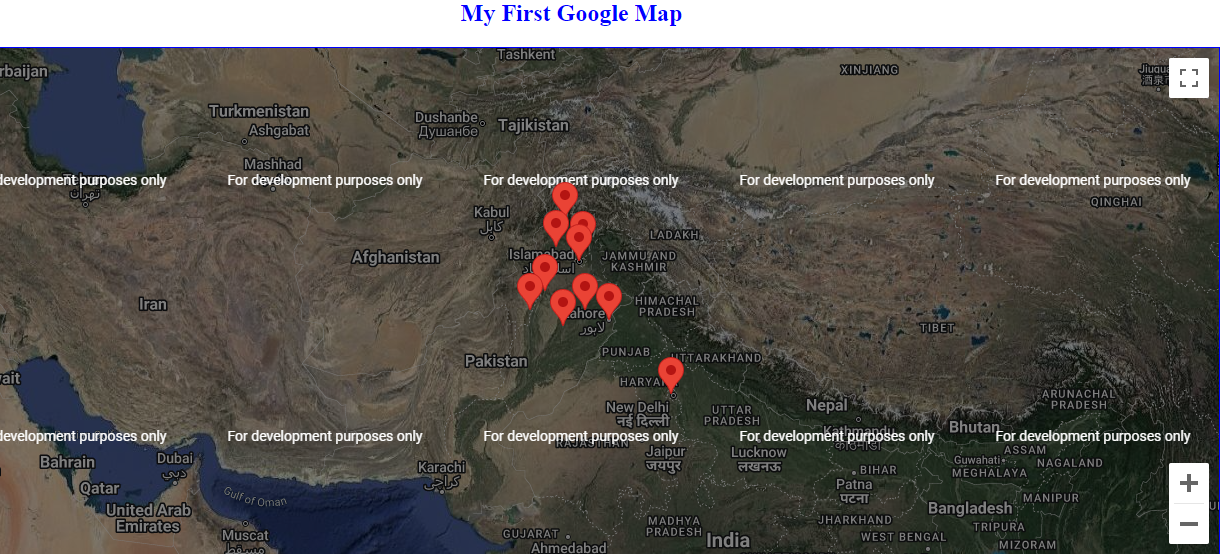
            map.setZoom(8);

          }, 5000);

        });

****



****