API Guide

Layer API

Create a layer

1. GeoJson Post the GeoJson file to the endpoint
    - endpoint: /layers.json
    - method: POST
    - params:
        - layer[name]: optional
        - layer[geo_file]: required
    - return: layers created in json
2. Shapefile Post the Shapefile file to the endpoint
    - endpoint: /layers/upload_shapefile.json
    - method: POST
    - params:
        - layer[name]: optional
        - layer[geo_file]: required
    - return: layers created in json
3. TopoJson Post the TopoJson file to the endpoint
    - endpoint: /layers/upload_topojson.json
    - method: POST
    - params:
        - layer[name]: optional
        - layer[upload]: required
    - return: layers created in json

Get layer

1. Get all layers Get all layers information in json format
    - endpoint: /layers.json
    - method: GET
    - return: layers in json
2. Get a layer Get a layer in json format
    - endpoint: /layers/:id.json
    - method: GET

- return: layer in json

Update a layer(only layer name now)

- endpoint: /layers/:id.json
- method: PUT
- params:
  - layer[name]: optional

Destroy a layer

- endpoint: /layers/:id.json
- method: DELETE

Point Query API

Query point

1. Query a point
   - endpoint: /points/:id,:lon,:lat
   - method: GET
   - return: {id => id, layer_id => layer_id, area_id => area_id, points => ({id => id, x =>x, y =>y}, {id => id, x =>x, y =>y}, …), :pointsWithinCount => xx}
2. Query multiple points
   - endpoint: /points/:id1,:lon1,:lat1;:id2,:lon2,:lat2
   - method: GET
   - return: {id => id, layer_id => layer_id, area_id => area_id, points => ({id => id, x =>x, y =>y}, {id => id, x =>x, y =>y}, …), :pointsWithinCount => xx}

Query point against layer

1. Query a point
   - endpoint: /layers/:layer_id/points/:id,:lon,:lat.json
   - method: GET
   - return: {id => id, layer_id => layer_id, area_id => area_id, points => ({id => id, x =>x, y =>y}, {id => id, x =>x, y =>y}, …), :pointsWithinCount => xx}
2. Query multiple points
   - endpoint: /layers/:layer_id/points/:id1,:lon1,:lat1;:id2,:lon2,:lat2.json

- method: GET
- return: {id => id, layer_id => layer_id, area_id => area_id, points => ({id => id, x =>x, y =>y}, {id => id, x =>x, y =>y}, …), :pointsWithinCount => xx}

Multi-data query(CSV file) API

- endpoint: /data/show.json
- method: POST
- params:
  - layer_id
  - csv_file(file to be uploaded)
- return:
  - data_json
  - permalink

High Performance Utility API

1. Query a count of points in given areas
   - endpoint: /api/query_points_count_of_areas
   - method: POST
   - params, hash with keys areas_ids, points, table_name
     - areas_ids: An array including all polygons you want to query
     - points: An array including all row_indexes of points you want to query
     - table_name: points table name (all points are saved to a temp db table when importing csv)
   - returns: A hash json with area_id as key and count as value {area_id1 => count1, area_id2 => count2}
2. Query a count of points in the polygons belonging to a given layer
   - endpoint: /api/query_points_count_of_layer_areas
   - method: POST
   - params: A hash with keys layer_id, points, table_name
     - layer_id: layer with the id includes some areas
     - points: An array including all row_indexes of points
     - table_name: points table name (all points are saved to a temp db table when importing csv)
   - return: A hash with area_id as key and count as value {area_id1 => count1, area_id2 => count2}

3. Find polygons and counts of points in polygon on a given layer (when in the multilevel layers, click one area to show the children)
    - endpoint: /api/find_layer_children
    - method: POST
    - params: A hash with keys layer_id, points, table_name
        - layer_id: The layer includes the area and can be used to judge is this level is penultimate
        - area_id: The area is used to query its children
        - table_name: points table name (all points are saved to a temp db table when importing csv)
    - return: A hash with children, counts, points, is_penultimate, layer_id { :children => {area_id1 => unproject_exterior_ring, ...}, :counts => {area_id1 => count1, area_id2 => count2 ...}, :points => {area_id1 => [p1, p2, ..], area_id2 => [p3, p4, ..]}, (points are in area) :is_penultimate => true/false, :layer_id => the id of given layer's child }
4. Find count of attributes belonging to one column
    - endpoint: /api/find_points_within_area
    - method: POST
    - params: hash with keys area_id, table_name
        - area_id: The area used to be query
        - table_name: points table name (all points are saved to a temp db table when importing csv)
    - return: An array including the indexes of points belonging to the area [p1, p2, p3 ...]
5. Find count of attributes belonging to one column
    - endpoint: /api/find_filters_info
    - method: POST
    - params: hash with keys row_indexes, col_name, table_name
        - row_indexes: The row_index of points
        - col_name: The column which is used to query the attrs with counts belonging to the column
        - table_name: points table name (all points are saved to a temp db table when importing csv)
    - return: A hash with counts attrs of points { :count_of_attrs => {attr1 => count1, attr2 => count2, ...} :attrs_of_points => {index_of_point1 => attr1, index_of_point2 => attr2} }
6. Find all attributes and values for points
    - endpoint: /api/find_point_info
    - method: POST
    - params: A hash with keys index, table_name

        - row_indexes: Row_index of the point
        - table_name: points table name (all points are saved to a temp db table when importing csv)
    - return: A hash with attribute as key, value as value { attr1 => value1, attr2 => value2, attr3 => value3 ... }