Application Functionality for :
 (A) The user chooses a point inside of either the T1,T2, and/or T3 isochrones. Returned are: Printed statement “You are within the service area of…”, distance to the facility and/or facilities that are contained within the isochrone(s), the type of facility/facilities, and name or names.

(B) The user chooses a point outside of any existing isochrone. Returned are: Printed statement, “You are outside the service area of any existing public health facility. The distance to the nearest medical facility is Xkm”, Distance to nearest facility in km, type of facility, and name

Distances:
T1 - 5km, 10km
T2 - 3km, 5km
T3 - 1km, 3km

Code

```
CREATE TABLE goa_mf_t1 AS
  SELECT *
  FROM goa_mf
  WHERE goa_mf.facility_t='dis_h';

 CREATE TABLE goa_mf_t2 AS
  SELECT *
  FROM goa_mf
  WHERE goa_mf.facility_t='chc' or goa_mf.facility_t='phc' or goa_mf.facility_t='s_t_h';

  CREATE TABLE goa_mf_t3 AS
  SELECT *
  FROM goa_mf
  WHERE goa_mf.facility_t='sub_cen';

#separating the facilities into 3 tiers

CREATE TABLE nearest_neighbors AS
SELECT hospitals.gid AS hospital_gid,
       hospitals.facility_n AS hospital,
       nodes.osm_id AS node,
       nodes.id AS node_gid,
       nodes.the_geom::geometry(Point, 4326) AS node_geom,
       nodes.dist
FROM goa_mf hospitals
CROSS JOIN LATERAL (
  SELECT nodes.osm_id, nodes.the_geom, nodes.id, nodes.the_geom <-> hospitals.geom AS dist
  FROM nodes AS nodes
  ORDER BY dist
  LIMIT 1
) nodes;

#identifies the nearest nodes to each facility

CREATE TABLE nearest_neighbors_t1 AS
SELECT hospitals.gid AS hospital_gid,
       hospitals.facility_n AS hospital,
       nodes.osm_id AS node,
       nodes.id AS node_gid,
       nodes.the_geom::geometry(Point, 4326) AS node_geom,
       nodes.dist
FROM goa_mf_t1 hospitals
CROSS JOIN LATERAL (
  SELECT nodes.osm_id, nodes.the_geom, nodes.id, nodes.the_geom <-> hospitals.geom
AS dist
  FROM nodes AS nodes
  ORDER BY dist
  LIMIT 1
) nodes;

  CREATE TABLE nearest_neighbors_t2 AS
SELECT hospitals.gid AS hospital_gid,
       hospitals.facility_n AS hospital,
       nodes.osm_id AS node,
       nodes.id AS node_gid,
       nodes.the_geom::geometry(Point, 4326) AS node_geom,
       nodes.dist
FROM goa_mf_t2 hospitals
CROSS JOIN LATERAL (
  SELECT nodes.osm_id, nodes.the_geom, nodes.id, nodes.the_geom <-> hospitals.geom
AS dist
  FROM nodes AS nodes
  ORDER BY dist
  LIMIT 1
) nodes;

CREATE TABLE nearest_neighbors_t3 AS
SELECT hospitals.gid AS hospital_gid,
       hospitals.facility_n AS hospital,
       nodes.osm_id AS node,
       nodes.id AS node_gid,
       nodes.the_geom::geometry(Point, 4326) AS node_geom,
       nodes.dist
FROM goa_mf_t3 hospitals
CROSS JOIN LATERAL (
  SELECT nodes.osm_id, nodes.the_geom, nodes.id, nodes.the_geom <-> hospitals.geom
AS dist
  FROM nodes AS nodes
  ORDER BY dist
  LIMIT 1
) nodes;

#joins the nearest nodes to each facilities into each tier


SELECT
h.facility_n AS hospital_id,
f.id AS node_id
INTO t1_nodes
FROM tier_1
AS h
CROSS
JOIN LATERAL
( SELECT nn.osm_id
as id, nn.the_geom
as geom
FROM nodes nn
ORDER BY (nn.the_geom <-> h.geom)
LIMIT
1)
AS f;

SELECT
h.facility_n AS hospital_id,
f.id AS node_id
INTO t2_nodes
FROM tier_2
AS h
CROSS
JOIN LATERAL
( SELECT nn.osm_id
as id, nn.the_geom
as geom
FROM nodes nn
ORDER BY (nn.the_geom <-> h.geom)
LIMIT
1)
AS f;

SELECT
h.facility_n AS hospital_id,
f.id AS node_id
INTO t3_nodes
```

```
FROM tier_3
AS h
CROSS
JOIN LATERAL
( SELECT nn.osm_id
as id, nn.the_geom
as geom
FROM nodes nn
ORDER BY (nn.the_geom <-> h.geom)
LIMIT
1)
AS f;

SELECT pgr_drivingDistance(
'SELECT gid as id, source, target, length_m as cost, length_m as reverse_cost
FROM network',
hn.node_gid, 10000
) as dd, hn.hospital_gid
FROM nearest_neighbors_t1 as hn

# calculates the pgrouting driving distance based on a specific radius around the central
point (the facility)

WITH driving_distance
AS (
SELECT hn.hospital_gid,
(pgr_drivingDistance('SELECT gid as id, source, target, length_m as cost, length_m as
reverse_cost FROM network',
hn.node_gid,
10000,
directed:=false)) AS result
from nearest_neighbors_t1
AS hn
)
SELECT
hospital_gid,
(result).node AS node_id,
(result).agg_cost AS distance
INTO nodes_distance_to_t1_yay
FROM driving_distance;

# embeds the driving distance function to calculate distances

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
```

```
into t1_isochrones
FROM
nodes_distance_to_t1_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
group by ndh.hospital_gid;

# draws the isochrone based on equal driving distance


SELECT pgr_drivingDistance(
'SELECT gid as id, source, target, length_m as cost, length_m as reverse_cost
FROM network',
hn.node_gid, 5000
) as dd, hn.hospital_gid
FROM nearest_neighbors_t2 as hn

WITH driving_distance
AS (
SELECT hn.hospital_gid,
(pgr_drivingDistance('SELECT gid as id, source, target, length_m as cost, length_m as
reverse_cost FROM network',
hn.node_gid,
5000,
directed:=false)) AS result
from nearest_neighbors_t2
AS hn
)
SELECT
hospital_gid,
(result).node AS node_id,
(result).agg_cost AS distance
INTO nodes_distance_to_t2_yay
FROM driving_distance;

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
into t2_isochrones
FROM
nodes_distance_to_t2_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
group by ndh.hospital_gid;
```

```
SELECT pgr_drivingDistance(
'SELECT gid as id, source, target, length_m as cost, length_m as reverse_cost
FROM network',
hn.node_gid, 3000
) as dd, hn.hospital_gid
FROM nearest_neighbors_t3 as hn

WITH driving_distance
AS (
SELECT hn.hospital_gid,
(pgr_drivingDistance('SELECT gid as id, source, target, length_m as cost, length_m as
reverse_cost FROM network',
hn.node_gid,
3000,
directed:=false)) AS result
from nearest_neighbors_t3
AS hn
)
SELECT
hospital_gid,
(result).node AS node_id,
(result).agg_cost AS distance
INTO nodes_distance_to_t3_yay
FROM driving_distance;

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
into t3_isochrones
FROM
nodes_distance_to_t3_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
group by ndh.hospital_gid;

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
into t1_isochrones_small
FROM
nodes_distance_to_t1_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
and distance < 5000
group by ndh.hospital_gid;

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
into t2_isochrones_small
FROM
nodes_distance_to_t2_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
and distance < 3000
group by ndh.hospital_gid;

SELECT
ndh.hospital_gid,
st_concavehull(st_collect(nn.the_geom),
0.9)
AS geom
into t3_isochrones_small
FROM
nodes_distance_to_t3_yay AS ndh,
nodes AS nn
WHERE
nn.id = ndh.node_id
and distance < 1000
group by ndh.hospital_gid;


CREATE view t1_intersection AS
select h.geom, c.tot_p, h.hospital_id
from t1_isochrones as h,
census_2001 as c
where st_intersects(h.geom, c.geom);

# step 1 of the intersection calculates the intersection of geometries between the census and medical facility layers

CREATE view t1_intersect_pop as
select h.hospital_id, sum(c.tot_p) as total_population
from t1_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
```

```
group by h.hospital_id;

# step 2 calculates a sum of the total population represented by all of the census tracts
intersected by the isochrones

CREATE view t1_intersect_pop_sum as
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t1_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

# step 3 calculates the area percentage of the census tracts covered by the isochrone and
then multiplies the that by the total population

CREATE view t1_intersection_small AS
select h.geom, c.tot_p, h.hospital_id
from t1_isochrones_small as h,
census_2001 as c
where st_intersects(h.geom, c.geom);

CREATE view t1_intersect_pop_small as
select h.hospital_id, sum(c.tot_p) as total_population
from t1_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t1_intersect_pop_sum_small as
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t1_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t2_intersection AS
select h.geom, c.tot_p, h.hospital_id
from t2_isochrones as h,
census_2001 as c
where st_intersects(h.geom, c.geom);

CREATE view t2_intersect_pop as
select h.hospital_id, sum(c.tot_p) as total_population
from t2_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t2_intersect_pop_sum as
```

```
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t2_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t2_intersection_small AS
select h.geom, c.tot_p, h.hospital_id
from t2_isochrones_small as h,
census_2001 as c
where st_intersects(h.geom, c.geom);

CREATE view t2_intersect_pop_small as
select h.hospital_id, sum(c.tot_p) as total_population
from t2_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t2_intersect_pop_sum_small as
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t2_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t3_intersection AS
select h.geom, c.tot_p, h.hospital_id
from t3_isochrones as h,
census_2001 as c
where st_intersects(h.geom, c.geom);

CREATE view t3_intersect_pop as
select h.hospital_id, sum(c.tot_p) as total_population
from t3_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t3_intersect_pop_sum as
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t3_isochrones as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t3_intersection_small AS
select h.geom, c.tot_p, h.hospital_id
from t3_isochrones_small as h,
census_2001 as c
```

```
where st_intersects(h.geom, c.geom);

CREATE view t3_intersect_pop_small as
select h.hospital_id, sum(c.tot_p) as total_population
from t3_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

CREATE view t3_intersect_pop_sum_small as
select h.hospital_id, sum(c.tot_p * st_area(st_intersection(h.geom, c.geom)) /
st_area(c.geom))
from t3_isochrones_small as h, census_2001 as c
where st_intersects(h.geom, c.geom)
group by h.hospital_id;

SELECT jsonb_build_object(
    'type',     'FeatureCollection',
    'features', jsonb_agg(feature)
)
FROM (
  SELECT jsonb_build_object(
    'type',       'Feature',
    'id',         hospital_gid,
    'geometry',   ST_AsGeoJSON(geom)::jsonb,
    'properties', to_jsonb(row) - 'hospital_gid' - 'geom'
  ) AS feature
  FROM (SELECT * FROM isochrones_all_joined) row) features;

# creates a feature collection from the geojson for all of the isochrones

SELECT jsonb_build_object(
    'type',     'FeatureCollection',
    'features', jsonb_agg(feature)
)
FROM (
  SELECT jsonb_build_object(
    'type',       'Feature',
    'id',         hospital_gid,
    'geometry',   ST_AsGeoJSON(hospital_geom)::jsonb,
    'properties', to_jsonb(row) - 'hospital_gid' - 'hospital_geom'
  ) AS feature
  FROM (SELECT * FROM hospitals_all_joined) row) features;

# does the same thing for all of the hospitals

 SELECT row_to_json(fc)
 FROM ( SELECT 'FeatureCollection' As type, array_to_json(array_agg(f)) As features
 FROM (SELECT 'Feature' As type
    , ST_AsGeoJSON(lg.geom)::json As geometry
    , row_to_json((SELECT l FROM (SELECT hospital_gid, estimated_population_served,
facility_n, isochrone_size, facility_t, size_in_km) As l
      )) As properties
   FROM isochrones_all_joined As lg   ) As f )  As fc;

# converts the geojson to json format readable by the web app for the isochrones

   SELECT row_to_json(fc)
 FROM ( SELECT 'FeatureCollection' As type, array_to_json(array_agg(f)) As features
 FROM (SELECT 'Feature' As type
    , ST_AsGeoJSON(lg.hospital_geom)::json As geometry
    , row_to_json((SELECT l FROM (SELECT hospital_gid, estimated_population_served,
facility_n, isochrone_size, facility_t, size_in_km) As l
      )) As properties
   FROM hospitals_all_joined As lg   ) As f )  As fc;

# same as above for the hospitals
```