package com.example.vrptruck;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

import org.apache.http.HttpEntity;

import org.apache.http.HttpResponse;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.impl.client.DefaultHttpClient;

import org.json.JSONArray;

import org.json.JSONObject;

import android.app.Activity;

import android.content.IntentSender;

import android.location.Location;

import android.os.Bundle;

import android.view.Menu;

import android.widget.Toast;

import com.google.android.gms.common.ConnectionResult;

import com.google.android.gms.common.GooglePlayServicesClient;

import com.google.android.gms.location.LocationClient;

import com.google.android.gms.location.LocationListener;

import com.google.android.gms.location.LocationRequest;

import com.google.android.gms.maps.CameraUpdateFactory;

import com.google.android.gms.maps.GoogleMap;

import com.google.android.gms.maps.MapFragment;

import com.google.android.gms.maps.model.CameraPosition;

import com.google.android.gms.maps.model.LatLng;

import com.google.android.gms.maps.model.MarkerOptions;

import com.google.android.gms.maps.model.Polyline;

import com.google.android.gms.maps.model.PolylineOptions;

public class VRPMap extends Activity implements GooglePlayServicesClient.ConnectionCallbacks,

GooglePlayServicesClient.OnConnectionFailedListener,

LocationListener{

private static final String TAG\_LAT="lat";

private static final String TAG\_LONG="lng";

private static final String TAG\_ROUTES="routes";

private static final String TAG\_LEGS="legs";

private static final String TAG\_STEPS="steps";

private static final String TAG\_END="end\_location";

List<LatLng> Loc\_temp = new ArrayList<LatLng>();

List<LatLng> Cust\_Loc = new ArrayList<LatLng>();

GoogleMap map;

JSONArray steps;

public static int cnt;

boolean flag=false;

private final static int

CONNECTION\_FAILURE\_RESOLUTION\_REQUEST = 9000;

LocationClient loc\_client;

// Milliseconds per second

private static final int MILLISECONDS\_PER\_SECOND = 1000;

// Update frequency in seconds

public static final int UPDATE\_INTERVAL\_IN\_SECONDS = 5\*60;

// Update frequency in milliseconds

private static final long UPDATE\_INTERVAL =

MILLISECONDS\_PER\_SECOND \* UPDATE\_INTERVAL\_IN\_SECONDS;

// The fastest update frequency, in seconds

private static final int FASTEST\_INTERVAL\_IN\_SECONDS = 1\*60;

// A fast frequency ceiling in milliseconds

private static final long FASTEST\_INTERVAL =

MILLISECONDS\_PER\_SECOND \* FASTEST\_INTERVAL\_IN\_SECONDS;

LocationRequest mLocationRequest;

double latitude,longitude;

double lat1,lat2,long1,long2;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.vrp\_map);

Cust\_Loc=UserPage.Loc;

map = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();

for(int i=1;i<Cust\_Loc.size();i++)

{

map.addMarker(new MarkerOptions()

.position(Cust\_Loc.get(i))

.title("Customer" + i));

}

loc\_client=new LocationClient(this, this, this);

mLocationRequest = LocationRequest.create();

mLocationRequest.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);

mLocationRequest.setInterval(UPDATE\_INTERVAL);

mLocationRequest.setFastestInterval(FASTEST\_INTERVAL);

new Thread(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

bounds(Cust\_Loc.get(0), Cust\_Loc.get(Cust\_Loc.size()-1));

for(cnt=0;cnt<Cust\_Loc.size()-1;cnt++)

{

parse(Cust\_Loc.get(cnt),Cust\_Loc.get(cnt+1));

}

}

}).start();

}

private void bounds(LatLng org, LatLng dest){

try{

String O = String.valueOf(org.latitude);

String D = String.valueOf(org.longitude);

String O1 = String.valueOf(dest.latitude);

String D1 = String.valueOf(dest.longitude);

HttpClient client = new DefaultHttpClient();

HttpPost post = new HttpPost("http://maps.googleapis.com/maps/api/directions/json?origin="+O+","+D+"&destination="+O1+","+D1+"&sensor=false&avoid=highways&mode=driving");

HttpResponse response = client.execute(post);

HttpEntity entity = response.getEntity();

InputStream is = entity.getContent();

BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"), 8);

StringBuilder sb = new StringBuilder();

String line = null;

String result;

while ((line = reader.readLine()) != null)

{

sb.append(line + "\n");

}

result = sb.toString();

//System.out.println("JSON" + result);

JSONObject json = new JSONObject(result);

JSONArray rt = json.getJSONArray(TAG\_ROUTES);

JSONObject temp1 = rt.getJSONObject(0);

JSONArray legs = temp1.getJSONArray(TAG\_LEGS);

JSONObject end\_loc = legs.getJSONObject(0);

JSONObject end1 = end\_loc.getJSONObject(TAG\_END);

final String l = end1.getString(TAG\_LAT);

final String lo = end1.getString(TAG\_LONG);

JSONObject start1 = end\_loc.getJSONObject("start\_location");

final String l1 = start1.getString(TAG\_LAT);

final String lo1 = start1.getString(TAG\_LONG);

lat1 = Double.parseDouble(l);

lat2=Double.parseDouble(l1);

long1=Double.parseDouble(lo);

long2=Double.parseDouble(lo1);

System.out.println("BOUNDS:"+ l+" "+lo+" &"+l1+" "+lo1);

}

catch(Exception e){

System.err.println("Bounds:"+e);

}

}

private void parse(LatLng org, LatLng dest)

{

try{

String O = String.valueOf(org.latitude);

String D = String.valueOf(org.longitude);

String O1 = String.valueOf(dest.latitude);

String D1 = String.valueOf(dest.longitude);

HttpClient client = new DefaultHttpClient();

HttpPost post = new HttpPost("http://maps.googleapis.com/maps/api/directions/json?origin="+O+","+D+"&destination="+O1+","+D1+"&sensor=false&avoid=highways&mode=driving");

HttpResponse response = client.execute(post);

HttpEntity entity = response.getEntity();

InputStream is = entity.getContent();

BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"), 8);

StringBuilder sb = new StringBuilder();

String line = null;

String result;

while ((line = reader.readLine()) != null)

{

sb.append(line + "\n");

}

result = sb.toString();

//System.out.println("JSON" + result);

JSONObject json = new JSONObject(result);

JSONArray rt = json.getJSONArray(TAG\_ROUTES);

int i=0,j=0,k=0;

JSONArray st=null,leg=null;

for( i = 0; i < rt.length(); i++){

JSONObject c = rt.getJSONObject(i);

leg = c.getJSONArray(TAG\_LEGS);

for( j=0;j<leg.length();j++){

JSONObject ob = leg.getJSONObject(j);

st = ob.getJSONArray(TAG\_STEPS);

for(k=0;k<st.length();k++)

{

JSONObject loc = st.getJSONObject(k);

JSONObject end = loc.getJSONObject(TAG\_END);

final String lat = end.getString(TAG\_LAT);

final String longitude = end.getString(TAG\_LONG);

runOnUiThread(new Runnable() {

public void run() {

try{

Double d1 = Double.parseDouble(lat);

Double d2 = Double.parseDouble(longitude);

LatLng l1 = new LatLng(d1,d2);

Loc\_temp.add(l1);

System.out.println(lat+" "+longitude+"\n");

}

catch(Exception e){

System.err.println("From UIThread" + e);

}

}

});

}

}

}

if( k>=st.length())

{flag=true;

System.out.println("FLAG:" + flag);}

runOnUiThread(new Runnable() {

public void run() {

try{

Draw\_PolyLine(Loc\_temp);

}

catch(Exception e){

System.err.println("From UIThread 2" + e);

}

}

});

}

catch(Exception e){

System.err.println(e);

}

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

private void Draw\_PolyLine(List<LatLng> Loc)

{

map.setMapType(GoogleMap.MAP\_TYPE\_NORMAL);

System.out.println("Inside Polyline");

map.addMarker(new MarkerOptions()

.position(Loc.get(0))

.title("Origin"));

CameraPosition cameraPosition = new CameraPosition.Builder()

.target(Loc.get(0)) // Sets the center of the map

.zoom(17) // Sets the zoom

.bearing(90) // Sets the orientation of the camera to east

.tilt(30) // Sets the tilt of the camera to 30 degrees

.build(); // Creates a CameraPosition from the builder

map.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));

// map.moveCamera(CameraUpdateFactory.newLatLngZoom(Loc.get(0), 10));

Polyline PL = map.addPolyline(new PolylineOptions()

.geodesic(false)

);

PL.setPoints(Loc);

geofence();

}

private void geofence(){

startPeriodicUpdates();

}

private void getLocation(){

Location loc;

loc = loc\_client.getLastLocation();

latitude = loc.getLatitude();

longitude=loc.getLongitude();

Toast.makeText(VRPMap.this, locationStringFromLocation(loc), Toast.LENGTH\_LONG).show();

}

private String locationStringFromLocation(final Location location) {

return Location.convert(location.getLatitude(), Location.FORMAT\_DEGREES) + " " + Location.convert(location.getLongitude(), Location.FORMAT\_DEGREES);

}

@Override

protected void onStart() {

super.onStart();

// Connect the client.

loc\_client.connect();

}

@Override

protected void onStop() {

if (loc\_client.isConnected()) {

stopPeriodicUpdates();

}

// Disconnecting the client invalidates it.

loc\_client.disconnect();

super.onStop();

}

@Override

public void onLocationChanged(Location location) {

// TODO Auto-generated method stub

String msg = "Updated Location: " +

Double.toString(location.getLatitude()) + "," +

Double.toString(location.getLongitude());

Toast.makeText(this, msg, Toast.LENGTH\_SHORT).show();

latitude = location.getLatitude();

longitude=location.getLongitude();

if(lat1>lat2)

{

if(!(latitude>(lat2+5) && latitude<(lat1+5)))

{

Toast.makeText(VRPMap.this, "Going outside permissible limits", Toast.LENGTH\_SHORT).show();

}

}

else

{

if((latitude>(lat2+5) && latitude<(lat1+5)))

{

Toast.makeText(VRPMap.this, "Going outside permissible limits", Toast.LENGTH\_SHORT).show();

}

}

if(long1>long2)

{

if(!(longitude>(long2+5) && longitude <(long1+5)) )

{

Toast.makeText(VRPMap.this, "Going outside permissible limits", Toast.LENGTH\_SHORT).show();

}

}

else{

if((longitude>(long2+5) && longitude <(long1+5)) )

{

Toast.makeText(VRPMap.this, "Going outside permissible limits", Toast.LENGTH\_SHORT).show();

}

}

}

private void startPeriodicUpdates() {

loc\_client.requestLocationUpdates(mLocationRequest, (LocationListener)this);

}

private void stopPeriodicUpdates() {

loc\_client.removeLocationUpdates(this);

}

@Override

public void onConnectionFailed(ConnectionResult result) {

// TODO Auto-generated method stub

if (result.hasResolution()) {

try {

// Start an Activity that tries to resolve the error

result.startResolutionForResult(

this,

CONNECTION\_FAILURE\_RESOLUTION\_REQUEST);

/\*

\* Thrown if Google Play services canceled the original

\* PendingIntent

\*/

} catch (IntentSender.SendIntentException e) {

// Log the error

e.printStackTrace();

}

} else {

/\*

\* If no resolution is available, display a dialog to the

\* user with the error.

\*/

Toast.makeText(VRPMap.this,result.getErrorCode() , Toast.LENGTH\_SHORT).show();

}

}

@Override

public void onConnected(Bundle connectionHint) {

// TODO Auto-generated method stub

Toast.makeText(VRPMap.this, "Connected", Toast.LENGTH\_LONG).show();

}

@Override

public void onDisconnected() {

// TODO Auto-generated method stub

Toast.makeText(VRPMap.this, "Dis-Connected", Toast.LENGTH\_LONG).show();

}

}