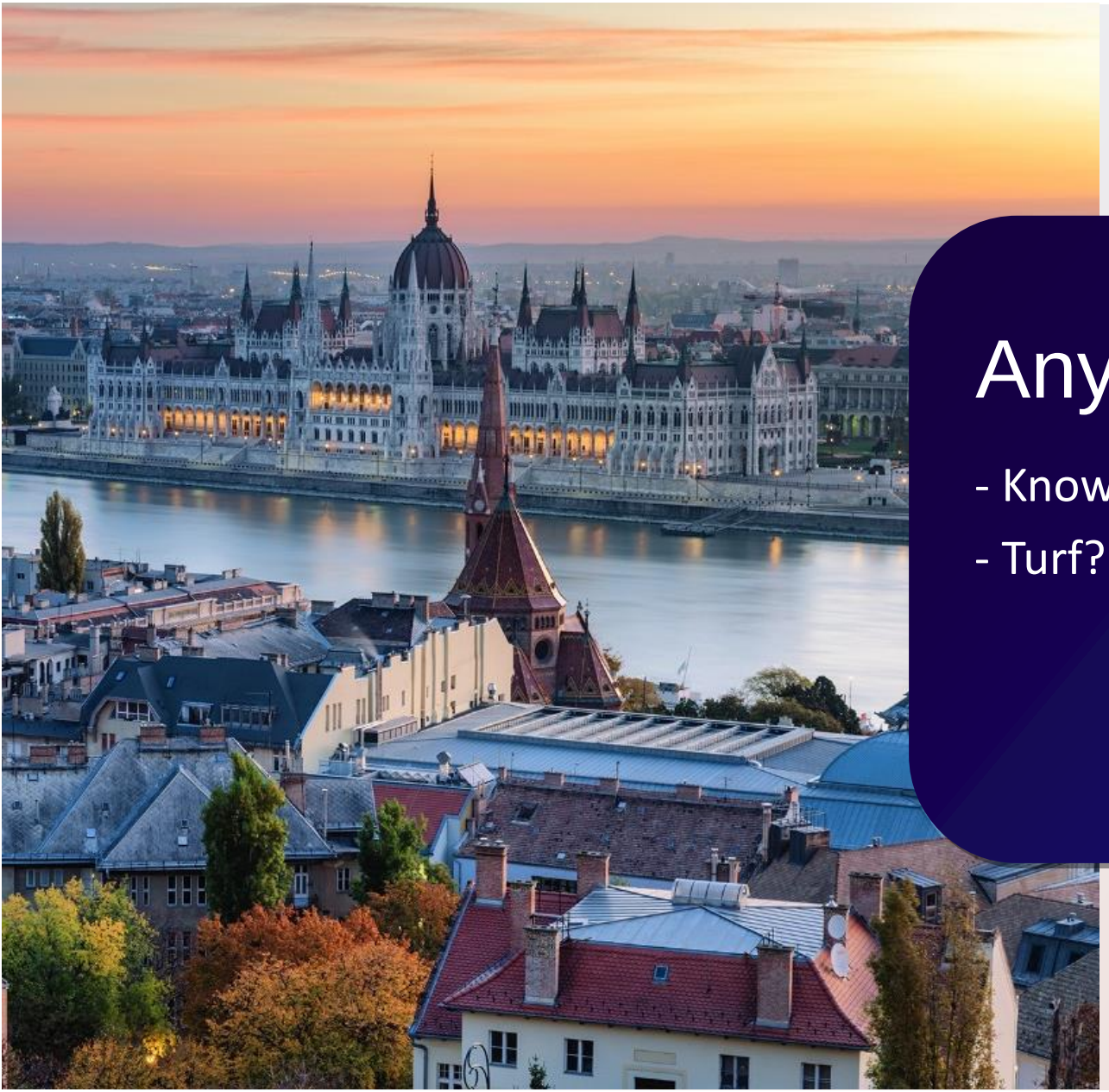
mndwrk

Geospatial modules in Node.js

Ákos Polgár

Why the hell should I use Node.js?

01



Any advantages?

- Knowledge of the team
- Turf?

mndwrk

MEASUREMENT

along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint -> only for 2 Points, no LineString
pointOnFeature
polygonTangents
pointToLineDistance
rhumbBearing
rhumbDestination
rhumbDistance
square
GreatCircle

FEATURE CONVERSION

combine
explode
flatten
lineToPolygon
polygonize
polygonToLine

COORDINATE MUTATION

cleanCoords
flip
rewind
round
truncate

TRANSFORMATION

bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset
polygonSmooth
simplify
tesselate
transformRotate
transformTranslate
transformScale
union
voronoi

mndwrk

MISC
ellipse
kinks
lineArc
lineChunk
lineIntersect
lineOverlap
lineSegment
lineSlice
lineSliceAlong
lineSplit
mask
nearestPointOnLine
sector
shortestPath
unkinkPolygon

HELPER
featureCollection
feature
geometryCollection
lineString
multiLineString
multiPoint
multiPolygon
point
Polygon

RANDOM
randomPosition
randomPoint
randomLineString

DATA
sample

INTERPOLATION
interpolate
isobands
isolines
planepoint
tin

JOINS
pointsWithinPolygon
tag

GRIDS
hexGrid
pointGrid
squareGrid
triangleGrid

CLASSIFICATION
nearestPoint

AGGREGATION
collect
clustersDbscan
ClustersKmeans

ASSERTIONS
collectionOf
containsNumber
geojsonType
featureOf

mndwrk

META

coordAll
coordEach
coordReduce
featureEach
featureReduce
flattenEach
flattenReduce
getCoord
getCoords
getGeom
getType
geomEach
geomReduce
propEach
propReduce
segmentEach
segmentReduce
getCluster
clusterEach
clusterReduce

BOOLEANS

booleanClockwise
booleanConcave
booleanContains
booleanCrosses
booleanDisjoint
booleanEqual
booleanIntersects
booleanOverlap
booleanParallel
booleanPointInPolygon
booleanPointOnLine
booleanWithin

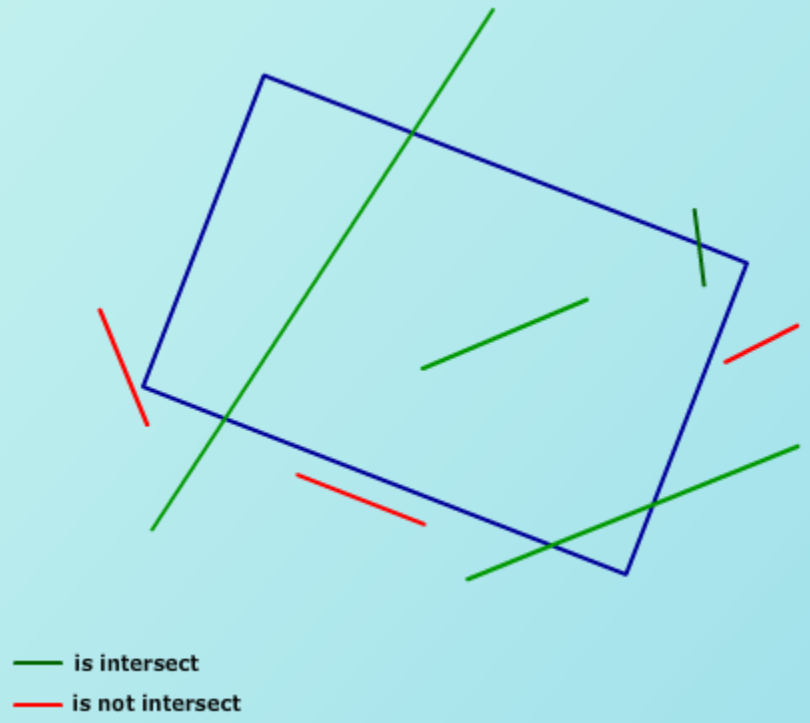
UNIT CONVERSION

bearingToAzimuth
convertArea
convertLength
degreesToRadians
lengthToRadians
lengthToDegrees
radiansToLength
radiansToDegrees
toMercator
toWgs84

Concrete issues

02

Eliminate lines by Polygons



Eliminate lines by Polygons

```
const basePath = require('base-path')('gdc-core');
const turf = require('turf');
const _ = require('lodash');
const myApp = require('@gis/line-slicer');
const {
    equalPoints
} = require(basePath.join('lib/helpers'));
const progress = require(basePath.join('lib/progress'));

exports.exec = function(conf) {
    var lineSections = conf.getInput('lineSections');
    var segmenterPolygons = conf.getInput('segmenterPolygons');
    var outerLineColl = conf.getOutput('cleanedLines');



        ...

};
```

Eliminate lines by Polygons

```
function findPolygons() {                                   //Reading data from mongo
    return segmenterPolygons.find({
        $or: [{
            "geometry.type": "MultiPolygon"
        }, {
            "geometry.type": "Polygon"
        }]
    }).toArray();
}

function findLines() {
    return lineSections.find({
        "geometry.type": "LineString"
    }).toArray();
}

function findRemainingObjects(coll) { //Which are not lineStrings
    return coll.find({
        "geometry.type": {              $ne: "LineString"           }
    }).toArray();
}
```

Eliminate lines by Polygons

```
return Promise.all([
        findPolygons(),
        findLines()
    ])
    .then((results) => {
        var [polygons, inputLines] = results;
        var outputLines = [];

        const progIndicator = progress.loop({
            total: polygons.length
        });

        _.forEach(polygons, (polygon) => {
            ...
    }).then(() => {
        return findRemainingObjects(lineSections);
    }).then((res) => {
        if (res.length > 0) {
            return outerLineColl.insertMany(res);
        }
    })
```

Eliminate lines by Polygons

```
.forEach(polygons, (polygon) => {
        outputLines = [];
        _.forEach(inputLines, (inputLine) => {
           var slices = myApp.lineSliceAtIntersection(inputLine, polygon);

           slices.features.forEach(function(ft) {

              ...
           });
        });
        progIndicator.progress();
        inputLines = outputLines;    //Otherwise next Polygon would process the original input
    });

    return outerLineColl.insertMany(outputLines);
```

Eliminate lines by Polygons


```
while (ft.geometry.coordinates.length > 1 && equalPoints(ft.geometry.coordinates[0], ft.geometry.coordinates[1])) {
        //Remove the first element if the first and second is identical
        ft.geometry.coordinates.shift();
    }

    if (ft.geometry.coordinates.length < 2) {
        return; //Wrong geometry
    }

    if (turf.lineDistance(ft.geometry) < 0.0001) { //10 cm, too short
        return;
    }
                                //Using euclidean way instead of topology -> quick, not puctual, but it's fair enough
    if (turf.inside(myApp.getMidPoint(ft.geometry), polygon)) {
        return; //Don't need these
    }
    ft.id = inputLine.id;
    outputLines.push(ft);
```

Change metadata of short lines

```
class ShortLineRemover extends GisModule {
   start() {
      const alterableColl = this.getOutput('alterableColl');

      lineLength = this.getOption('length') / 1000;
      return findLines(alterableColl, this.getCityCode())
      .then((lines) => {
         var loopLines = lines.map(line => changeShortLines(alterableColl, line)); // run the function over all items.
         return Promise.all(loopLines);
      });
   }
}

module.exports = new ShortLineRemover(config);
```

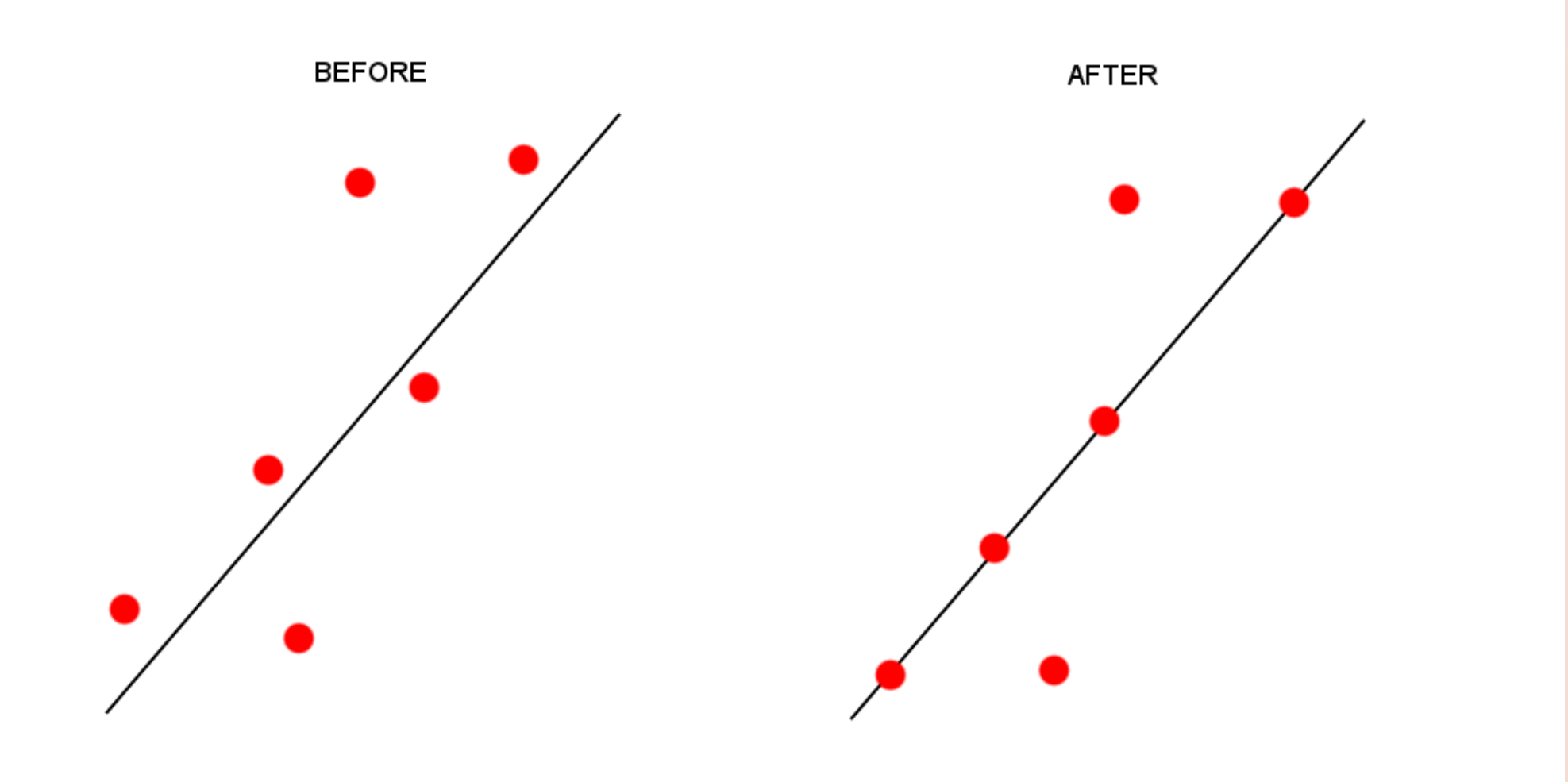
Change metadata of short lines

```
function findLines(targetColl) {                    // Load lines from DB
   let cond = {
      $and: [
         {"geometry.type": "LineString"},
         { $where: "this.conditions.length < 2" },    //It has only 1 condition
         {"conditions.type":"UNREGULATED"}          //which is UNREGULATED
      ]};
   return targetColl.find(cond).toArray();
}

function changeShortLines(targetColl, line) {
   if (turf.lineDistance(line.geometry, 'kilometers') < lineLength) {
      return targetColl.update({
         _id: line._id
      },{ $set: {
         "originalRuleName.en" : "No Parking",
         "conditions": [ { "type": "NO_SPACE" } ]
      }});
   }

   return Promise.resolve();
}
```

Align points to lines



Point to closest line alignment

```
function findClosestLinesToPoint(collection, docPoint, distance) {
   return collection.find({
      geometry: {
         $near: {
            $geometry: {
               type: "Point",
               coordinates: docPoint.geometry.coordinates
            },
            $maxDistance: Number(distance)
         }
      }
   }).toArray();
}
```

Point to lines alignment

```
return inputCollection.find({
      "geometry.type": "Point"
   }).toArray()
      .then((docPoints) => {
         return Promise.each(docPoints, (doc) => {
            var pt = turf.point(doc.geometry.coordinates);
            var against = {
               "type": "FeatureCollection",
               "features": []
            };

            return findClosestLinesToPoint(shifterColl, doc, distance)
               .then((closestLines) => {
                                    ...
               })..then(() => {
                                                ...
                                    });
         })
      })
```

Point to closest line alignment

```
.then((closestLines) => {
    closestLines.forEach((geom) => {
        if (geom.geometry.type == "LineString") {
            var line = turf.lineString(geom.geometry.coordinates);
            var snapped = turf.pointOnLine(line, pt, 'miles'); //calculates the closest Point on the LineString.
            snapped.properties.lineId = geom.id;
            against.features.push(snapped);              // Store all closest points of all closest lines into against array
        }
    });
}).then(() => {
    if (!against.features.length) { return; }
    const closestPoint = turf.nearest(pt, against);

    return inputCollection.update({
        '_id': doc._id
    }, {
        $set: {
            "geometry.coordinates": closestPoint.geometry.coordinates,
            "isShifted": true
        }
    }, { upsert: true });
});
```

Parallel processing - parent

```
const child_process = require('child_process');
const Promise = require("bluebird");
const mongoskin = require('mongoskin');
const numchild = require('os').cpus().length;
const db = mongoskin.db("mongodb://localhost:27019/my_database");
const INPUT_POINTS = db.collection("parkingMeters");
const INPUT_LINES = db.collection("rawSections_2018_03_15");

var done = 0;
readAllPointsFromMongodb().then(function (allPoints) {
      var arrayOfArrays = chunkify(allPoints, numchild);
      for (var i = 0; i < numchild; i++){
         var child = child_process.fork('./child');                    //path of the module
         child.send(arrayOfArrays[i]);
         child.on('message', function() {
            done++;  //parent received message from child;
            if (done === numchild) {
               db.close();    //parent received all results
            }
         });
      }
   });
```

Parallel processing - parent

```
function chunkify(a, n) {    // https://stackoverflow.com/questions/8188548/splitting-a-js-array-into-n-arrays
   if (n < 2)
      return [a];

   var len = a.length,
      out = [],
      i = 0,
      size;

   if (len % n === 0) {
      size = Math.floor(len / n);
      while (i < len) {
         out.push(a.slice(i, i += size));
      }
   } else {
      while (i < len) {
         size = Math.ceil((len - i) / n--);
         out.push(a.slice(i, i += size));
      }
   }
   return out;
}
```

Parallel processing - child

```
const turf = require('turf');
const Promise = require("bluebird");
const mongoose = require('mongoose');
const mongoskin = require('mongoskin');
const db = mongoskin.db("mongodb://localhost:27019/my_database");
const INPUT_LINES = db.collection("rawSections_2018_03_15");
const distance = 5;
const kpsResult = db.collection("shiftedPoints");
var globalRes = [];
```

Parallel processing - child

```
function findClosestLinesToPoint(docPoint) {
    return new Promise(function (resolve, reject) {
        INPUT_LINES.find({
            $and: [{
                "geometry.type": "LineString"
            }, {
                geometry: {
                    $near: {
                        $geometry: {
                            type: "Point",
                            coordinates: docPoint.geometry.coordinates
                        },
                        $maxDistance: Number(distance)
                    }
                }
            }]
        }).toArray(function (err, docsLines) {
            if (err) {
                reject();
            }
            resolve(docsLines);
        });    });   }
```

Parallel processing - child

```
process.on('message', function(allDocPoints) {
   return Promise.each(allDocPoints, (docPoint) => {
      docPoint._id = mongoose.Types.ObjectId(docPoint._id);
      var pt = turf.point(docPoint.geometry.coordinates);
      var against = {
         "type": "FeatureCollection",
         "features": []
      };
      return findClosestLinesToPoint(docPoint)
         .then(function (closestLines) {
            if (!closestLines) {
               return Promise.resolve(docPoint);
            }
            closestLines.forEach(function (geom) {          //find the closest point on a closestLine
               var line = turf.lineString(geom.geometry.coordinates);
               var snapped = turf.pointOnLine(line, pt, 'miles');    //Takes a pt and a line and calculates the closest pt on the line
               snapped.lineId = geom._id;
               against.features.push(snapped);
            });    ...
```

Parallel processing - child

```
...
            if (against.features.length !== 0) {
               docPoint.geometry = turf.nearest(pt, against).geometry;
               docPoint.properties.lineId = turf.nearest(pt, against).lineId;
               docPoint.properties.isShifted = true;
            }
            return Promise.resolve(docPoint);
         })
         .then(function (docPoint) {
            globalRes.push(docPoint);
         })
   }).then(function () {
            kpsResult.insert(globalRes, function (err) {
               if (err)   {                  console.log("error is :" + err);   }
               process.send({
                  child   : process.pid,
                  result  : allDocPoints.length + 1
               });
               db.close();
               process.disconnect();
            });
})      });
```

Q&A

apolgar.dev@gmail.com

www.facebook.com/vegtelenvakacio