```
/**
    This code was imported and modified to create our hyperbolic tree-map
    Radial Reingold–Tilford Tree
    http://bl.ocks.org/mbostock/4063550
*/

function drawTreeMap(svgTree){
    var diameter = window.innerWidth-dodWindowSize; // the width of svgTree
    var height = window.innerHeight;
    var jsonText="";//string that contains json data
    var jsonObject=null;
    var treeFiles = [];
    var treeRoot=null;


    // puts data in tree structure, store in jsonText as a string
    MakeTree(curr);

    // converts string to json object
    jsonObject=JSON.parse(jsonText);

    // removes previous tree if there was one
    svgTree.selectAll("g").remove();

    // creates tree layout
    var tree = d3.layout.tree()
        .size([360, 300])
        .separation(function(a, b) { return (a.parent == b.parent ? 1 : 2) / a.depth; });

    // creates new diagonal generator
    var diagonal = d3.svg.diagonal.radial()
        .projection(function(d) { return [d.y, d.x / 180 * Math.PI]; });

    // moves tree to the center of the svg
    var svg = svgTree.append("g")
        .attr("transform", "translate(" + diameter / 2 + "," + window.innerHeight/ 2 + ")");

    // selects the blank area in the svg so users can click anywhere to pan and zoom
    svgTree.select(".boundingRect").attr("fill", "white");

    // handling zooming and panning
    svgTree.call(d3.behavior.zoom()
        .translate([0,0])
        .scale(1)
        .scaleExtent([0.1, 8.0])
        .on("zoom", function(){
            svgTree.select("g").attr("transform", "scale("+d3.event.scale+") "+"translate(" + (
            diameter/2+d3.event.translate[0]) + "," + (window.innerHeight/2+d3.event.translate[1
            ]) + ")");
        }));

    // extracts data from jsonObject and put into node and link arrays
    var nodes = tree.nodes(jsonObject),
        links = tree.links(nodes);

    // tree links
    var link = svg.selectAll(".link")
        .data(links)
    .enter().append("path")
        .attr("class", "link")
        .attr("d", diagonal);

    // tree nodes
    treeNodes = svg.selectAll(".node")
        .data(nodes)
    .enter().append("g")
        .attr("class", "node")
        .attr("transform", function(d) { return "rotate(" + (d.x - 90) + ")translate(" + d.y +
        ")"; })


    //  draw nodes
    treeNodes.each(function(d) {

        // draws folder nodes
        if(d.size==null){
            var rect = d3.select(this).append("rect")
                .attr("width", 8)
                .attr("height", 8)
                .attr("x", -4)
                .attr("y", -4)
                .attr("fid", d)
                // create new tree using this as root
                .on("click", function(f){
                    setCurr(f.fid);
                });
            d.element = rect; // marks data as rectangle

        // draws file nodes
        } else {
            var tempCat = typeToCat[d.type];
            if(tempCat==null) tempCat="Other";
            var circle = d3.select(this).append("circle")
                .attr("r", 3)
                .attr("type", d.type)
                .style("fill", catToColor[tempCat])
                .style("stroke", "black")
                .style("stroke-width", .5);

            d.element = circle; // marks data as circle
        }
    });

    treeNodes.each(function(d) {
        // gets only the name of the root folder
        var slashSplit = d.name.split("/");
        if (slashSplit.length>1) d.name = slashSplit[slashSplit.length-1];

        var isFolder = d.size==null;

        // shows text only in certain depths
        if(!isFolder || d.depth<3){

            // draws nodes' name
            var text = d3.select(this).append("text")
                .attr("dy", ".31em")
                .attr("text-anchor", function(d) { return d.x < 180 ? "start" : "end"; })
                .attr("transform", function(d) { return d.x < 180 ? "translate(8)" :
                "rotate(180)translate(-8)"; })
                .text(function(d) { return d.name; });

            // shows file name on mouseover of circle
            if (!isFolder){
                text.attr("display", "none");
                d.element.on("mouseover", function(){
                    text.attr("display", null);
                    console.log(d);
                    var file = files[d.fid];
                    refreshFileDetailsOnDemand(file);
                });
                d.element.on("mouseout", function(){
                    text.attr("display", "none");
                    refreshFolderDetailsOnDemand(curr);
                });
            }
            d.text = text;
        }
    });

    d3.select(self.frameElement).style("height", diameter - 150 + "px");


    // extracts data from treeFiles array and store it in jsonText in json format (as tree
    structure)
    function MakeTree(currFolder){
        // the beginning of current folder
        jsonText+="{"+JSON.stringify("name")+":"+JSON.stringify(currFolder.name);
        jsonText+=","+JSON.stringify("fid")+":"+JSON.stringify(currFolder.fid);

        // the end of current folder
        if(currFolder.fileChildren.length==0 && currFolder.folderChildren.length==0)
            jsonText+="}";

        // if current folder has children
        if(currFolder.fileChildren.length>0 || currFolder.folderChildren.length>0)
            jsonText+=","+JSON.stringify("children")+": [";

        // extracts data from file children array
        for (var i=0; i<currFolder.fileChildren.length; i++){
            jsonText+=JSON.stringify({
                name: currFolder.fileChildren[i].name,
                size: currFolder.fileChildren[i].size,
                type: currFolder.fileChildren[i].type,
                fid: currFolder.fileChildren[i].fid
            });

            // if current folder has more than one file child
            if(i!=currFolder.fileChildren.length-1)
                jsonText+=",";
        }

        // if current folder has any file or folder child
        if(currFolder.folderChildren.length>0 && currFolder.fileChildren.length>0){
            jsonText+=",";
        }

        // extracts data from folder children array
        for (var i=0; i<currFolder.folderChildren.length; i++){

                // calls MakeTree on folder children
                MakeTree(currFolder.folderChildren[i]);

                // if current folder has more than one folder child
                if(i!=currFolder.folderChildren.length-1)
                    jsonText+=",";
        }

        // the end of file/folder children array
        if(currFolder.fileChildren.length>0 || currFolder.folderChildren.length>0)
            jsonText+="]}";

    }

}
```