

Create PT

index.tsx

```
import {start} from "../express/resolve"

start()
```

express/resolve.tsx

```
// I built this myself (you can see it on npmjs)
// Built by [redacted] (tnichols217) at https://www.npmjs.com/package/basicjsx
import React from "basicjsx"
// Express http server for running the web server
// Made by TJ Holowaychuk, can be found at https://www.npmjs.com/package/express
import express, { RequestHandler, Request, Response } from "express";
// Http for using the http protocol (because theres no need for https for this project)
// Built in module for the node.js environment
import http from "http";
import { resolve as main } from "../routes/main"

//@ts-ignore
import monke from "../routes/monke.glb"
//@ts-ignore
import hdri from "../routes/hdri.exr"

const app = express()

export type dirList = string[]
export type resolveTreeFunction = (dir: dirList, q: any, res: Response) => void
export type resolveTreeItem = resolveTreeFunction | resolveTree
export type resolveTree = [resolveTreeFunction, { [key: string]: resolveTreeItem }]

const resTreeTup = {
  Func: 0,
  ResTreeItem: 1
} as const

const sendFile: resolveTreeFunction = (req, q, res) => {
  if (req.length == 1) {
    console.log(req)
    res.sendFile(`${__dirname}/${req[0]}`)
  } else {
    res.send("Failed to find file")
  }
}

const mainWrapper: resolveTreeFunction = (req, q, res) => {
  console.log(req)
  if (req.length == 0) {
    return main(req, q, res)
  } else {
    console.log(req)
    return sendFile(req, q, res)
  }
}
```

```

}

const examples: resolveTreeFunction = (req, q, res) => {
  const obj = ".glb"
  const env = ".exr"
  let path = `${__dirname}/files/`
  let file = ""
  let ext = ""
  console.log(req)
  if (req.length == 0) {
    file = "monke"
    ext = obj
  } else if (req.length == 1) {
    return res.redirect(`./${req[0]}/view`)
  } else {
    if (req[1] == "view") {
      return main(req, q, res)
    }
    file = req[0]
    ext = req[1] == "env" ? env : obj
  }
  path += file + ext
  res.sendFile(path)
}

const mapDirs: resolveTree = [
  mainWrapper,
  {
    obj: (req, q, res) => {
      res.sendFile(`${__dirname}/${monke}`)
    },
    env: (req, q, res) => {
      res.sendFile(`${__dirname}/${hdri}`)
    },
    examples: examples
  }
]

const resolveDir = async (dir: dirList, mD: resolveTree = mapDirs) => {
  return new Promise<[resolveTreeFunction, dirList]>((resolve, reject) => {
    if (dir.length == 0) {
      return resolve([mD[resTreeTup.Func], []])
    }
    let newDir = mD[resTreeTup.ResTreeItem][dir[0]]
    let newPath = dir.slice(1)
    if (newDir == null) {
      return resolve([mD[resTreeTup.Func], dir])
    }
    if (typeof newDir == "function") {
      return resolve([newDir, newPath])
    } else {
      return resolveDir(newPath, newDir).then(resolve).catch(reject)
    }
  })
}

const resolve = async (req: Request, res: Response) => {
  let args = req.params['0'].split("/").slice(1)
  let q = req.query

  if (args.length > 1 && args[args.length - 1] == "") {

```

```

    res.redirect(req.params['0'].slice(0, -2))
    return
  } else if (args.length == 1 && args[0] == "") {
    args = []
  }

  resolveDir(args).then(([func, args]) => func(args, q, res)).catch(console.error)
}

export const start = () => {
  app.get("*", resolve)
  http.createServer(app).listen(8080)
}

```

express/routes/main.tsx

```

import { resolveTreeFunction } from "../resolve"
// I built this package
// Built by [redacted] (tnichols217) at https://www.npmjs.com/package/basicjsx
import React from "basicjsx"
import { CustomElements } from "basicjsx"
//@ts-ignore
import main from "./main.client"
//@ts-ignore
import css from "./main.css"

export const wrapFunction = (func: any, args : string) => {
  console.log(func)
  return `(${func.toString()})(${args})`
}

const JS = (props, children) => {
  let str = `(${props.js.toString()})()`
  delete props.js
  let out = React.createElement("script", props, [str as any])
  return out
}

const OBJ = (props, children) => {
  let str = JSON.stringify(props.json)
  return str
}

const ImportMap = {
  "imports": {
    "three": "https://unpkg.com/three@0.151.3/build/three.module.js",
    "three/addons/": "https://unpkg.com/three@0.151.3/examples/jsm/",
    "postprocessing":
      "https://unpkg.com/postprocessing@6.30.2/build/postprocessing.mjs",
    "realism-effects": "https://unpkg.com/realism-effects@1.0.19/dist/index.js"
  }
}

//enable polyfill if importmaps arent available
const Head = () =>
<head>
  { /* <script async src="https://ga.jspm.io/npm:es-module-shims@1.7.1/dist/es-module-shims.js"></script> */ }
  <script type="importmap">

```

```

        {JSON.stringify(ImportMap)}
      </script>
      <style>
        {css}
      </style>
    </head>

    const Body = () => <body>
      <script type="module">
        {Buffer.from(main, 'base64').toString()}
      </script>
      <div id="drop_zone" ondrop="window.dropHandler(event);"
ondragover="window.dragOverHandler(event);">
      </div>
    </body>

    export const resolve: resolveTreeFunction = (dir, query, res) => {
      let ret = <html>
        <Head />
        <Body />
      </html>
      res.send("<!DOCTYPE html>" + ret.outerHTML)
    }

```

express/routes/main.client.ts

```

/* IMPORTS */

// Three.js for starting the 3d canvas view
// Built by mrdoob, at https://www.npmjs.com/package/three
import * as THREE from 'three';
// For loading GLTF files (for displaying)
import {GLTFLoader} from 'three/addons/loaders/GLTFLoader';
// For loading EXR files (for background and luminosity)
import {EXRLoader} from 'three/addons/loaders/EXRLoader';
// For basic controls orbiting around the model
import {OrbitControls} from 'three/addons/controls/OrbitControls.js';
// Postprocessing to allow filters and processing of the canvas
// Built by mrdoob and vanruesc at https://www.npmjs.com/package/postprocessing
import * as POSTPROCESSING from "postprocessing"
// realism-effects to enable Motion Blur
// made by 0beqz at https://github.com/0beqz/realism-effects
import { MotionBlurEffect, VelocityDepthNormalPass } from "realism-effects"

/* INIT */
const camera = new THREE.PerspectiveCamera(70, window.innerWidth / window.innerHeight,
0.01, 10);
camera.position.z = 1;

const scene = new THREE.Scene();

/* LOADERS */
let gltfLoader = new GLTFLoader();
let exrLoader = new EXRLoader();

let [gltf, ext] = [
  gltfLoader.loadAsync('./obj'),
  exrLoader.loadAsync("./env"),
]

```

```

/* SCENE */

let gltfFile

gltf.then((gltf) => {
    gltfFile = gltf.scene;
    scene.add(gltfFile);
});

/* RENDERER */

const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setAnimationLoop(animation);
document.getElementById("drop_zone").appendChild(renderer.domElement);
renderer.toneMapping = THREE.ACESFilmicToneMapping;
renderer.outputEncoding = THREE.sRGBEncoding;
renderer.shadowMap.enabled = true

const composer = new POSTPROCESSING.EffectComposer(renderer)

// EFFECTS
const renderPass = new POSTPROCESSING.RenderPass( scene, camera )
composer.addPass( renderPass )
const velocityDepthNormalPass = new VelocityDepthNormalPass(scene, camera)
composer.addPass(velocityDepthNormalPass)

const motionBlurEffect = new MotionBlurEffect(velocityDepthNormalPass)
const effectPass = new POSTPROCESSING.EffectPass(camera, motionBlurEffect)
composer.addPass(effectPass)

composer.setSize(window.innerWidth, window.innerHeight);

const pmremGenerator = new THREE.PMREMGenerator(renderer);
pmremGenerator.compileEquirectangularShader();

/* BACKGROUND */

ext.then((texture) => {
    texture.mapping = THREE.EquirectangularReflectionMapping;
    scene.environment = texture;
    scene.background = texture;

    texture.dispose();
});

/* SCENE CONTROLS */

const controls = new OrbitControls(camera, renderer.domElement);
controls.target.set(0, 0, 0);
controls.update();

/* ANIMATION */

function resizeRendererToDisplaySize(renderer, camera) {
    const canvas = renderer.domElement;
    const width = window.innerWidth;
    const height = window.innerHeight;

```

```

const needResize = canvas.width !== width || canvas.height !== height;
if (needResize) {
    renderer.setSize(width, height);
    composer.setSize(width, height);
    const canvas = renderer.domElement;
    camera.aspect = canvas.clientWidth / canvas.clientHeight;
    camera.updateProjectionMatrix();
}
return needResize;
}

function animation(time) {

    resizeRendererToDisplaySize(renderer, camera)

    // renderer.render(scene, camera);
    composer.render();
}

/* DRAG AND DROP HANDLERS */

window.dropHandler = function (event) {
    let backgroundExt = [".exr"]
    let glExt = [".glb"]

    event.preventDefault();
    let dt = Array.from(event.dataTransfer.items) ?? []
    dt = dt.filter((item) => item.kind == "file")

    let newBack = dt.find((item) => backgroundExt.filter((ext) =>
item.getAsFile().name.endsWith(ext)).length > 0)
    let newGl = dt.find((item) => glExt.filter((ext) =>
item.getAsFile().name.endsWith(ext)).length > 0)

    if (newGl) {
        scene.remove(gltfFile)
        newGl.getAsFile().arrayBuffer().then((AB) => {
            gltfLoader.parse(AB, "", (gl) => {
                gltfFile = gl.scene;
                scene.add(gltfFile);
            })
        })
    }

    if (newBack) {
        newBack.getAsFile().arrayBuffer().then((AB) => {
            let texData = exrLoader.parse(AB)

            // Manually parse new texture for webGl, since there is no buffer
loader for EXR files

            const texture = new THREE.DataTexture();

            let usedKeys = ["encoding", "format", "type"]
            let imageKeys = ["width", "height", "data"]
            let texKeys = Object.entries(texData)
            Object.assign(texture,
                Object.fromEntries(texKeys.filter(([key, _]) => {
                    return usedKeys.includes(key)
                })),

```

```

        {
            image: Object.fromEntries(texKeys.filter(([key, _])
=> {
                return imageKeys.includes(key)
            })),
            magFilter: THREE.LinearFilter,
            minFilter: THREE.LinearFilter,
            needsUpdate:true
        }
    )

    texture.mapping = THREE.EquirectangularReflectionMapping;
    scene.environment = texture;
    scene.background = texture;

    texture.dispose();
}
})
}

window.dragOverHandler = function (event) {
    event.preventDefault();
}

```

express/routes/main.css

```

* {
    margin: 0;
    padding: 0;
}

```

monke.glb

This is a gltf 3d model file, for loading the 3d model in the viewer

hdri.exr

This is an exr 360 environment file for loading the background in the scene