```
{% load static %}
const grafo = require("./modules/grafo");
const readline = require("readline");
const express = require("express");
const path = require("path");
const fs = require("fs");
const app = express();

const http = require("http").createServer(app);
const webPort = 8080;

const pathDiccionario = path.join(
  __dirname,
  "..",
  "archivos",
  "diccionario.txt"
);
const pathPrettyTree = path.join(__dirname, "..", "archivos", "tree.json");
const pathCounter = path.join(__dirname, "..", "archivos", "contador.txt");
const pathProceso = path.join(__dirname, "..", "archivos", "proceso.txt");
const pathChart = path.join(__dirname, "..", "archivos", "chart.json");
const pathInput = path.join(__dirname, "..", "archivos", "input.txt");

let treantTree = {};
let counter = {};
let q0;
let io;

// EJS INIT
// Set the view engine to ejs
app.set("view engine", "ejs");
// Set ejs files path
app.set("views", __dirname + "/../dist/pages");

// REQUESTS
app.get("/", (req, res) => {
  res.render("index");
});
app.get("/get/tree/", (req, res) => {
  res.end(JSON.stringify(treantTree));
});
app.get("/generate/tree", (req, res) => {
  var response = {
    valid: false,
    message: "Archvio no encontrado!",
  };

  // Check if it exists
  try {
    if (fs.existsSync(pathDiccionario)) {
      response.valid = true;
      response.message = "Archivo cargado!";
    }
  } catch (err) {}
```

```
55
56    res.end(JSON.stringify(response));
57
58    generateTree();
59    generateTreantTree();
60  });
61  app.get("/start/", (req, res) => {
62    res.end(
63      JSON.stringify({
64        start: true,
65      })
66    );
67    processFile();
68  });
69
70  // NODE CREATOR
71  // Recursively adds a node
72  function addNode(node, word, value) {
73    let nodeName = value.substring(0, value.length - (word.length - 1));
74
75    if (word.length == 1) {
76      node.children[word] = new grafo.QNode(value, word, true);
77    } else {
78      // If child does not exist, create it.
79      if (typeof node.children[word[0]] == "undefined") {
80        node.children[word[0]] = new grafo.QNode(
81          nodeName,
82          word.substring(0, 1)
83        );
84      }
85
86      // Add children node
87      node.children[word[0]] = addNode(
88        node.children[word[0]],
89        word.substring(1),
90        value
91      );
92    }
93    return node;
94  }
95
96  // MAIN AUTOMATA
97  async function processFile() {
98    const readable = readline.createInterface({
99      input: fs.createReadStream(pathInput, { encoding: "utf8" }),
100   });
101   const writeStream = fs.createWriteStream(pathProceso);
102   let currentNodes = [q0];
103   let nextNodes;
104   let progress;
105   let message;
106   let limi;
107
108   console.log(`Reading file ${pathInput}...`);
```

```
109
110   for await (let line of readable) {
111     // Add space to end of the line
112     line += " ";
113     limi = line.length;
114
115     for (let i = 0; i < limi; ++i) {
116       // Empty needed arrays
117       nextNodes = [];
118       message = [];
119
120       // Reset progress
121       progress = `(${line[i]})=>\n`;
122
123       // Process all steps at the 'same time'
124       currentNodes.forEach((node) => {
125         // Add q0 to everything
126         nextNodes.push(q0);
127
128         // Add the rest of nodes
129         nextNodes.push(
130           node.process(line[i], q0, (node, char, next) => {
131             // Append progress to variable
132             progress += `\t${node.name} -> ${next}\n`;
133
134             // If it's an end node, add to counter
135             if (node.end) {
136               // If counter is nonexistant, create it
137               if (typeof counter[node.name] == "undefined") {
138                 counter[node.name] = 0;
139               }
140
141               // Add to counter
142               ++counter[node.name];
143
144               // Register counter
145               progress += `\t\t${node.name} +1\n`;
146             }
147
148             // Add to GUI list
149             message.push({
150               currentChar: char,
151               nodeIsEnd: node.end,
152               nodeStart: node.name,
153               nodeEnd: next,
154             });
155           })
156         );
157       });
158
159       // Write current step to web GUI
160       io.sockets.emit("update", JSON.stringify(message));
161
162       // Update counter on file
```

```
163         writeCounter();
164
165         // Write progress to file
166         writeStream.write(progress);
167
168         // Set the next nodes to be processed while also removing duplicate nodes
169         currentNodes = [...new Set(nextNodes)];
170       }
171     }
172
173     console.log("Done reading file!");
174     writeStream.close();
175   }
176   function removeDuplicateNodes(nodeArray) {
177     nodeArray.forEach((nodeArray) => {});
178   }
179   function writeCounter() {
180     fs.writeFile(pathCounter, JSON.stringify(counter, null, 4), function () {});
181   }
182
183   // TREE FUNCTIONALITY
184   async function generateTree() {
185     const readable = readline.createInterface({
186       input: fs.createReadStream(pathDiccionario, { encoding: "utf8" }),
187     });
188     let fileTree;
189
190     console.log("Generating tree from " + pathDiccionario);
191
192     // Generate origin node
193     q0 = new grafo.QNode("q0", "\0");
194
195     // Create a node line for each word
196     for await (let line of readable) {
197       q0 = addNode(q0, line, line);
198     }
199
200     fileTree = fs.createWriteStream(pathPrettyTree);
201     fileTree.write(JSON.stringify(q0, null, 4));
202     fileTree.close();
203   }
204   function generateTreantTree() {
205     treantTree.chart = JSON.parse(fs.readFileSync(pathChart));
206
207     treantTree.nodeStructure = getGraphNode(q0);
208   }
209
210   function getGraphNode(node) {
211     let result = {};
212     let children = [];
213     result.text = {
214       name: node.name,
215     };
216     if (node.end) {
```

```
217       result.HTMLclass = "node-end";
218     }
219
220     for (let [key, val] of Object.entries(node.children)) {
221       children.push(getGraphNode(val));
222     }
223
224     result.children = children;
225     return result;
226   }
227
228   // INITIALIZERS
229   function initSocket() {
230     io = require("socket.io")(http);
231
232     // WEB SOCKET
233     io.on("connection", (socket) => {
234       var data = {
235         event: "handshake",
236         data: "Hola! C:",
237       };
238
239       console.log("Conexi n a socket!");
240       socket.emit("handshake", JSON.stringify(data));
241     });
242
243     console.log(io.path());
244
245     console.log(`Socket ready on ${http}`);
246   }
247   async function initTrees() {
248     // READ TREE SOURCE FILE
249     await generateTree();
250     generateTreantTree();
251     processFile();
252   }
253
254   // SERVER SET-UP
255   app.use(express.static(__dirname + "/../dist/public/"));
256
257   // INITIALIZE THINGS
258   initSocket();
259   initTrees();
260
261   // SERVER LISTEN INIT
262   http.listen(webPort, () => {
263     console.log("Listening on port: " + webPort);
264   });
```