```
const xlsxFile = require("read-excel-file/node");
const bodyParser = require("body-parser");
const readline = require("readline");
const express = require("express");
const aNode = require("./aNode.js");
const path = require("path");
const fs = require("fs");
const app = express();

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

const excelPath = path.join(__dirname, "archivos", "ajedrez.xlsx");

var transitionTree;
var stack = [];

// EJS INIT
// Set the view engine to ejs
app.set("view engine", "ejs");
// Set ejs files path
app.set("views", __dirname + "/dist/pages");
// Set body parser
app.use(bodyParser.urlencoded({ extended: true }));

// REQUESTS
app.get("/", (req, res) => {
  res.render("index");
});

app.post("/procesar/cadena/", (req, res) => {
  let response = {};
  let output;

  if (req.body.input.length == 0) {
    req.body.input = generateBin(10);
  }

  output = automata(req.body.input);

  console.log(output);

  response = output.response;
  response.binario = req.body.input;

  fs.writeFile("./archivos/proceso.txt", output.file, () => {});

  // Send response
  res.send(JSON.stringify(response));
});

function generateBin(len) {
  let resultado = "";
  for (let i = 0; i < len; ++i) {
```

```
55      resultado += Math.floor(Math.random() * 2);
56    }
57    return resultado;
58  }
59
60  // AUTOMATA
61  // Main automata functionality
62  function automata(cadena) {
63    let limi = cadena.length;
64    let current, next;
65    let response = {
66      animations: [],
67    };
68    let result = {
69      file: "",
70    };
71
72    stack = [];
73    stack.push("F");
74
75    current = [startNode];
76    response.animations.push({
77      nodeName: current[0].name,
78      input: cadena,
79      stack: stack.join(""),
80    });
81
82    result.file += `d(${current[0].name}, ${cadena}, ${stack.join("")})
        `;
83    for (let i = 0; i < limi; ++i) {
84      next = [];
85      current.forEach((nodo) => {
86        next = next.concat(
87          nodo.evaluateChar(cadena[i], stack[stack.length - 1])
88        );
89      });
90
91      // Removing duplicates
92      current = [...new Set(next)];
93
94      // Set animation output
95      response.animations.push({
96        nodeName: current[0].name,
97        input:
98          cadena.substring(i + 1).length == 0
99            ? "e"
100           : cadena.substring(i + 1),
101       stack: stack.join(""),
102     });
103
104     //Set file output
105     result.file += `->\nd(${current[0].name}, ${cadena.substring(
106       i
107     )}, ${stack.join("")})`;
```

```
108     }
109
110     // Process remaining stack things
111     limi = stack.length;
112     for (let i = 0; i < limi; ++i) {
113       next = [];
114       current.forEach((nodo) => {
115         next = next.concat(nodo.evaluateChar("  ", stack[i]));
116       });
117
118       // Removing duplicates
119       current = [...new Set(next)];
120
121       // Set animation output
122       response.animations.push({
123         nodeName: current[0].name,
124         input: "e",
125         stack: stack.join("").substring(i + 1),
126       });
127
128       //Set file output
129       result.file += `->\nd(${current[0].name}, e, ${stack.join("")})`;
130     }
131
132     if (current[0].name == "qf") {
133       response.result = "La cadena es v lida!"
134     } else {
135       response.result = "La cadena no es v lida :c"
136     }
137
138     // Add animation list
139     result.response = response;
140
141     return result;
142   }
143
144   // Create automata
145   function initAutomata() {
146     startNode = new aNode("q0");
147     let q1 = new aNode("q1");
148     let q2 = new aNode("qf");
149
150     q2.callback = (char) => {
151       // Input was valid
152       console.log("Input was valid");
153     };
154
155     q1.nextNodes.push({
156       input: "  ",
157       stack: "F",
158       node: q2,
159       callback: () => {},
160     });
161     q1.nextNodes.push({
```

```
    input: "1",
    stack: "  ",
    node: q1,
    callback: () => {
      // Pop from stack
      stack.pop();
    },
  });

  startNode.nextNodes.push({
    input: "0",
    stack: "  ",
    node: startNode,
    callback: () => {
      // Push 'a' to stack
      stack.push("a");
    },
  });
  startNode.nextNodes.push({
    input: "1",
    stack: "  ",
    node: q1,
    callback: () => {
      // Pop from stack
      stack.pop();
    },
  });
}

// SERVER SET-UP
app.use(express.static(__dirname + "/dist/public/"));

// SERVER LISTEN INIT
http.listen(webPort, () => {
  console.log("Listening on port: " + webPort);
});

// Init
initAutomata();
```

```

@media only screen and (min-width: 768px) and (max-width: 991px) {

  #main {
    width: 712px;
    padding: 100px 28px 120px;
  }

  /* .mono {
    font-size: 90%;
  } */

  .cssbtn a {
    margin-top: 10px;
```

```
    margin-bottom: 10px;
    width: 60px;
    height: 60px;
    font-size: 28px;
    line-height: 62px;
  }
```

```
class TelgramRequestHandler(object):
    def handle(self):
        addr = self.client_address[0]          # Client IP-adress
        telgram = self.request.recv(1024)      # Recieve telgram
        print "From: %s, Received: %s" % (addr, telgram)
        return
```

```

@media only screen and (min-width: 768px) and (max-width: 991px) {

  #main {
    width: 712px;
    padding: 100px 28px 120px;
  }

  /* .mono {
    font-size: 90%;
  } */

  .cssbtn a {
    margin-top: 10px;
    margin-bottom: 10px;
    width: 60px;
    height: 60px;
    font-size: 28px;
    line-height: 62px;
  }
```