Packages

axios package – to get data from public server

required packages – axios body-parser express ejs

type: module

index.js

import axios from "axios";

import express from "express";

import bodyParser from "body-parser";

*const* app = express();

*let* response = await axios.get("https://bored-api.appbrewery.com/random");

*let* data = response.data;

app.use(bodyParser.urlencoded({extended:true}))

app.set("view engine","ejs")

*const* PORT = 3000;

app.get("/", (*req*,*res*)*=>*{

*res*.render("data",{data});

})

app.post("/activity",async (*req*,*res*)*=>*{

   try{

    response = await axios.get(`https://bored-api.appbrewery.com/${(*req*.body.participants!="random" && *req*.body.type!="random")?`filter?type=${*req*.body.type}&participants=${*req*.body.participants}`:(*req*.body.participants=="random" && *req*.body.type!="random")?`filter?type=${*req*.body.type}`:(*req*.body.participants!="random" && *req*.body.type=="random")?`filter?participants=${*req*.body.participants}`:"random"}`);

    data = *Array*.isArray(response.data)?response.data[Math.floor(Math.random() \* response.data.length)]:response.data;

   }catch(err){

    data = err.response.data.error;

   }

*res*.redirect("/");

})

app.listen(PORT,()*=>*{

    console.log("Server is running at ",PORT)

})

data.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <style>

        @import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap');

        \*{

*margin*: 0;

*padding*: 0;

*font-family*: 'Poppins', sans-serif;

        }

        body{

*background-color*: #fff9ee;

*display*: flex;

*flex-direction*: column;

*justify-content*: center;

*align-items*: center;

*height*: 100vh;

*row-gap*: 0.5rem;

        }

        select{

*background-color*: orange;

*color*: white;

*border*: none;

*border-radius*: 0.5rem;

*padding*: 0.2rem;

        }

        .data{

*border*:2px solid orange;

*padding*: 0.5rem;

*border-radius*: 0.5rem;

*display*: flex;

*flex-direction*: column;

*row-gap*: 1rem;

        }

        .data p{

*color*: #2c3e50;

        }

        .data span{

*color*: #7f8c8d;

        }

        .data div{

*display*: flex;

*justify-content*: space-between;

        }

    </style>

</head>

<body>

    <form action="/activity" method="POST">

        <select name="type" id="">

           <option value="random">Random type</option>

           <option value="education">Educational</option>

           <option value="charity">Charity</option>

           <option value="recreational">Recreational</option>

           <option value="relaxation">Relaxation</option>

           <option value="busywork">Busywork</option>

           <option value="social">Social</option>

           <option value="diy">Diy</option>

           <option value="music">Music</option>

        </select>

        <select name="participants" id="">

            <option value="random">Any number of people</option>

            <option value="1">1</option>

            <option value="2">2</option>

            <option value="3">3</option>

            <option value="4">4</option>

        </select>

        <input type="submit" value="Go">

    </form>

    <% if(locals.data){ %>

        <div class="data">

            <% if(typeof data === "string"){ %>

                <p><%= data %></p>

                <%} else {%>

                    <p><%= data.activity %></p>

            <div><span><%= data.type %></span><span>Participants: <%= data.participants %></span></div>

            <% } %>

        </div>

    <%} else {%>

        <p>No Activity found</p>

    <%}%>

</body>

</html>

API Authentication:

Types: No Authentication – do not required to authenticate

Basic Authentication – uses username and password (less secure)

API key Authentication – uses api key (secure)

Token based Authentication – uses token (more secure)

Index.js – All API requests covered

import axios from "axios";

import express from "express";

import bodyParser from "body-parser";

*const* app = express();

app.use(bodyParser.urlencoded({extended:true}))

app.set("view engine","ejs")

*const* PORT = 3000;

app.get("/", async (*req*,*res*)*=>*{

        try{

*const* response = await axios.get("https://secrets-api.appbrewery.com/user-secrets",{

            headers:{

                Authorization: 'Bearer 8502d63d-3757-4f49-89a9-e2836773d1fa'

            }

        })

*res*.send(response.data)

        }catch(error){

*res*.status(404).send("Error: "+error.message)

        }

})

app.get("/basicAuth",async (*req*,*res*)*=>*{

    try{

*const* response = await axios.get("https://secrets-api.appbrewery.com/random",{

        auth:{

            username:"shivabhai",

            password:"123456789"

        }

    })

*res*.send(response.data)

    }catch(error){

*res*.status(404).send("Error: "+error.message)

    }

})

app.get("/apiAuth",async (*req*,*res*)*=>*{

    try{

*const* response = await axios.get("https://secrets-api.appbrewery.com/filter",{

            params:{

                score:5,

                apiKey:"3f9fd037-ff0e-4f79-b791-2653f5df5483"

            }

        })

*res*.send(response.data)

    }catch(err){

*res*.status(404).send("Error: "+err.message)

    }

})

app.get("/tokenAuth",async (*req*,*res*)*=>*{

    try{

*const* response = await axios.get("https://secrets-api.appbrewery.com/secrets/1",{

            headers:{

                Authorization: "Bearer 8502d63d-3757-4f49-89a9-e2836773d1fa"

            }

        })

*res*.send(response.data)

    }catch(err){

*res*.on("error",(*error*)*=>*{

          console.log(*error*)

        })

    }

})

app.listen(PORT,()*=>*{

    console.log("Server is running at ",PORT)

})

API creation

import bodyParser from "body-parser";

import express from "express";

import axios from "axios";

import helmet from "helmet";

import morgan from "morgan";

import dotenv from "dotenv";

dotenv.config();

*const* employees= [

    {

      "id": 1,

      "name": "John Doe",

      "position": "Software Engineer",

      "department": "Engineering",

      "salary": 80000

    },

    {

      "id": 2,

      "name": "Jane Smith",

      "position": "Marketing Specialist",

      "department": "Marketing",

      "salary": 60000

    },

    {

      "id": 3,

      "name": "Mike Johnson",

      "position": "Financial Analyst",

      "department": "Finance",

      "salary": 70000

    },

    {

      "id": 4,

      "name": "Emily Davis",

      "position": "Human Resources Manager",

      "department": "Human Resources",

      "salary": 75000

    },

    {

      "id": 5,

      "name": "Alex Rodriguez",

      "position": "Sales Representative",

      "department": "Sales",

      "salary": 65000

    },

    {

      "id": 6,

      "name": "Sara Brown",

      "position": "Customer Support Specialist",

      "department": "Customer Support",

      "salary": 55000

    },

    {

      "id": 7,

      "name": "Chris Taylor",

      "position": "Product Manager",

      "department": "Product Management",

      "salary": 90000

    },

    {

      "id": 8,

      "name": "Amanda White",

      "position": "Quality Assurance Engineer",

      "department": "Engineering",

      "salary": 82000

    },

    {

      "id": 9,

      "name": "Mark Johnson",

      "position": "Sales Manager",

      "department": "Sales",

      "salary": 75000

    },

    {

      "id": 10,

      "name": "Olivia Davis",

      "position": "Customer Relations Specialist",

      "department": "Customer Support",

      "salary": 55000

    },

    {

      "id": 11,

      "name": "Brian Miller",

      "position": "Operations Analyst",

      "department": "Operations",

      "salary": 72000

    },

    {

      "id": 12,

      "name": "Jessica Garcia",

      "position": "Marketing Manager",

      "department": "Marketing",

      "salary": 90000

    },

    {

      "id": 13,

      "name": "Daniel Martinez",

      "position": "IT Support Specialist",

      "department": "IT",

      "salary": 68000

    },

    {

      "id": 14,

      "name": "Sophia Johnson",

      "position": "Product Designer",

      "department": "Product Management",

      "salary": 78000

    },

    {

      "id": 15,

      "name": "William Brown",

      "position": "Financial Planner",

      "department": "Finance",

      "salary": 70000

    },

    {

      "id": 16,

      "name": "Ella Taylor",

      "position": "Human Resources Specialist",

      "department": "Human Resources",

      "salary": 75000

    },

    {

      "id": 17,

      "name": "James Anderson",

      "position": "Customer Success Manager",

      "department": "Customer Support",

      "salary": 82000

    },

    {

      "id": 18,

      "name": "Grace Smith",

      "position": "Sales Associate",

      "department": "Sales",

      "salary": 65000

    },

    {

      "id": 19,

      "name": "Benjamin Davis",

      "position": "Software Developer",

      "department": "Engineering",

      "salary": 80000

    },

    {

      "id": 20,

      "name": "Ava Rodriguez",

      "position": "Product Owner",

      "department": "Product Management",

      "salary": 95000

    }

  ];

*let* response; // To send response

*const* app = express();

*const* PORT = process.env.PORT;

app.use(helmet()); // To protecting Data

app.use(morgan("combined")); // For connecting API's public

app.use(bodyParser.urlencoded({extended:true})) // To get data coming in

*let* filteredEmployees;

*let* employeeId;

*let* employeeData;

*let* employeeIndex;

*const* filterMiddleware = (*req*,*res*,*next*) *=>* {

    try{

        if(*req*.query.department && *req*.query.position && *req*.query.salary)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.department==*req*.query.department && *employee*.position==*req*.query.position && *employee*.salary==*req*.query.salary))

        else if(*req*.query.department && *req*.query.position)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.department==*req*.query.department && *employee*.position==*req*.query.position))

        else if(*req*.query.position && *req*.query.salary)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.position==*req*.query.position && *employee*.salary==*req*.query.salary))

        else if(*req*.query.department && *req*.query.salary)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.department==*req*.query.department && *employee*.salary==*req*.query.salary))

        else if(*req*.query.department)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.department==*req*.query.department))

        else if(*req*.query.position)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.position==*req*.query.position))

        else if(*req*.query.salary)

        filteredEmployees = employees.filter((*employee*) *=>* (*employee*.salary==*req*.query.salary))

        else throw {"error":"Specify filters like department,salary,etc."};

        if(filteredEmployees.length==0)

        throw {"error":"Don't have any data"};

        next();

    }catch(error){

*res*.send(error)

    }

}

*let* getEmployeeMiddleware = (*req*,*res*,*next*) *=>* {

    try{

        employeeId =*req*.params.id;

        if(!*Number*.isInteger(*Number*.parseInt(employeeId)))

        throw {"error":"Specify Id only"};

        employeeData = employees.find((*employee*)*=>* *employee*.id==employeeId)

        if(!employeeData)

        throw {"error":`Data not exists with given id ${employeeId}`};

        next();

    }catch(error){

*res*.send(error)

    }

}

*let* postEmployeeMiddleware = (*req*,*res*,*next*) *=>* {

   try{

    if(!(*req*.body.name && *req*.body.position && *req*.body.department && *req*.body.salary))

    throw {"error":"Incomplete details"};

    if(!*Number*.isInteger(*Number*.parseInt(*req*.body.salary)))

    throw {"error":"Salary cannot be in the form of string"};

    next();

   }catch(error){

*res*.send(error)

   }

}

*let* putEmployeeMiddleware = (*req*,*res*,*next*) *=>* {

    try{

        employeeId =*req*.params.id;

        if(!*Number*.isInteger(*Number*.parseInt(employeeId)))

        throw {"error":"Specify Id only"};

        employeeIndex = employees.findIndex((*employee*)*=>* *employee*.id==employeeId)

        if(employeeIndex==-1)

        throw {"error":"Data not found"};

        if(*req*.body.salary && !*Number*.isInteger(*Number*.parseInt(*req*.body.salary)))

        throw {"error":"Salary cannot be in the form of string"};

        employeeData = employees[employeeIndex];

        next()

    }catch(error){

*res*.send(error)

    }

}

*let* deleteEmployeeMiddleware = (*req*,*res*,*next*) *=>* {

    try{

        employeeId =*req*.params.id;

        if(!*Number*.isInteger(*Number*.parseInt(employeeId)))

        throw {"error":"Specify Id only"};

        employeeIndex = employees.findIndex((*employee*)*=>* *employee*.id==employeeId)

        if(employeeIndex==-1)

        throw {"error":`Data not exists with given id ${employeeId}. hence data cannot be deleted`};

        next();

    }catch(error){

*res*.send(error)

    }

}

app.get("/",(*req*,*res*)*=>*{

*res*.send(employees)

})

app.get("/random",(*req*,*res*)*=>*{

    employeeData = employees[Math.floor(Math.random()\*employees.length)];

*res*.send(employeeData)

})

app.get("/filter",filterMiddleware,(*req*,*res*)*=>*{

*res*.send(filteredEmployees)

})

app.get("/:id",getEmployeeMiddleware,(*req*,*res*)*=>*{

    res.send(employeeData)

})

app.post("/employee",postEmployeeMiddleware,(*req*,*res*)*=>*{

    employeeData = {

        "id":employees.length + 1,

        "name":req.body.name,

        "position":req.body.position,

        "department":req.body.department,

        "salary":req.body.salary

    }

    employees.push(employeeData)

    res.send(employees[employees.length-1])

})

app.put("/employee/:id",putEmployeeMiddleware,(*req*,*res*)*=>*{

    employeeData.name = req.body.name || employeeData.name;

    employeeData.position = req.body.position || employeeData.position;

    employeeData.department = req.body.department || employeeData.department;

    employeeData.salary = Number.parseInt(req.body.salary) || employeeData.salary;

    employees[employeeIndex] = employeeData;

    res.send(employees[employeeIndex])

})

app.patch("/employee/:id",putEmployeeMiddleware,(*req*,*res*)*=>*{

    employeeData.name = req.body.name || employeeData.name;

    employeeData.position = req.body.position || employeeData.position;

    employeeData.department = req.body.department || employeeData.department;

    employeeData.salary = Number.parseInt(req.body.salary) || employeeData.salary;

    employees[employeeIndex] = employeeData;

    res.send(employees[employeeIndex])

})

app.delete("/employee/:id",deleteEmployeeMiddleware,(*req*,*res*)*=>*{

    employeeData = employees.splice(employeeIndex,1)

    res.send(employeeData)

})

app.listen(PORT,(*req*,*res*)*=>*{

    console.log(`Server is running on port ${PORT}`)

})