/\*\*

\* Function that is called during the JavaScript Task execution.

\* @param {IntegrationEvent} event

\*/

function executeScript(event) {

let input = event.getParameter('message\_to\_claims\_ops');

let jsonOutput = {

message: {

Other\_Carrier\_Insurance: "No",

Member\_is\_calpers: "Yes",

claim\_details: {

"Claim Type": null,

"Claim id": null,

"Service From Date": null,

"Service Thru Date": null,

"total allowed amount": null,

"total charge amount": null

},

"Member Name": null,

"Member ID": null,

"Member DOB": null,

Action: "Routed to claims ops"

}

};

if (typeof input === 'string') {

// Define regex patterns

const claimTypeMatch = input.match(/Claim Type:\s\*([A-Za-z]+)/i);

const claimIdMatch = input.match(/Claim(?: ID| id):\s\*([\w\d]+)/i); // capture both numeric or alphanumeric

const serviceFromDateMatch = input.match(/Service From Date:\s\*([\d.]+)/i);

const serviceThruDateMatch = input.match(/Service Thru Date:\s\*([\d.]+)/i);

const allowedAmountMatch = input.match(/Total Allowed Amount:\s\*([\d.]+)/i);

const chargeAmountMatch = input.match(/Total Charge Amount:\s\*([\d.]+)/i);

const memberNameMatch = input.match(/Member name is\s\*([A-Za-z\s]+)/i);

const memberIdMatch = input.match(/Member ID[:\s]\*([\d]+)/i);

const dobMatch = input.match(/DOB[:\s]\*(\d{1,2}\/\d{1,2}\/\d{2})/i);

// Populate extracted values

if (claimTypeMatch) jsonOutput.message.claim\_details["Claim Type"] = claimTypeMatch[1];

if (claimIdMatch) jsonOutput.message.claim\_details["Claim id"] = claimIdMatch[1];

if (serviceFromDateMatch) jsonOutput.message.claim\_details["Service From Date"] = serviceFromDateMatch[1].replace('.', '');

if (serviceThruDateMatch) jsonOutput.message.claim\_details["Service Thru Date"] = serviceThruDateMatch[1].replace('.', '');

if (allowedAmountMatch) jsonOutput.message.claim\_details["total allowed amount"] = parseFloat(allowedAmountMatch[1]).toFixed(2);

if (chargeAmountMatch) jsonOutput.message.claim\_details["total charge amount"] = parseFloat(chargeAmountMatch[1]).toFixed(2);

if (memberNameMatch) jsonOutput.message["Member Name"] = memberNameMatch[1].trim();

if (memberIdMatch) jsonOutput.message["Member ID"] = memberIdMatch[1].trim();

// Validate DOB format (MM/DD/YY)

if (dobMatch) {

const dob = dobMatch[1];

const dobParts = dob.split('/');

if (dobParts.length === 3 && dobParts[0].length <= 2 && dobParts[1].length <= 2 && dobParts[2].length === 2) {

jsonOutput.message["Member DOB"] = dob;

} else {

throw new Error(`Invalid DOB format. Expected MM/DD/YY, got: ${dob}`);

}

}

// Detect Other Carrier Insurance

if (/do not have|does not have/i.test(input)) {

jsonOutput.message.Other\_Carrier\_Insurance = "No";

} else {

jsonOutput.message.Other\_Carrier\_Insurance = "Yes";

}

// Detect Member Calpers

if (/is calpers/i.test(input)) {

jsonOutput.message.Member\_is\_calpers = "Yes";

} else if (/is not calpers/i.test(input)) {

jsonOutput.message.Member\_is\_calpers = "No";

}

} else {

throw new Error('Input is not a valid string');

}

// Function to remove nulls

function removeNulls(obj) {

for (const key in obj) {

if (obj[key] === null || obj[key] === undefined) {

delete obj[key];

} else if (typeof obj[key] === 'object') {

removeNulls(obj[key]);

}

}

}

// Final clean-up

removeNulls(jsonOutput);

// Set final output

event.setParameter('output', JSON.stringify(jsonOutput));

}