/\*\*

\* Function called during the JavaScript Task execution

\* @param {IntegrationEvent} event

\*/

function executeScript(event) {

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

// Clean the input: Remove line breaks, carriage returns, trim spaces

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

input = input.replace(/\n/g, ' ').replace(/\r/g, ' ').trim();

} else {

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

}

// Initialize JSON structure

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"

}

};

// Regex patterns

const claimTypeMatch = input.match(/Claim Type:\s\*(\w+)/i);

const claimIdMatch = input.match(/Claim[ \_]?id[:\s]\*([\w\d]+)/i);

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

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

const allowedAmountMatch = input.match(/total allowed amount[:\s]\*(\d+\.?\d\*)/i);

const chargeAmountMatch = input.match(/total charge amount[:\s]\*(\d+\.?\d\*)/i);

const memberNameMatch = input.match(/member name is\s\*(.+?),/i);

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

const dobMatch = input.match(/DOB[:\s]\*(\d{1,2}\/\d{1,2}\/\d{2,4})/i);

// Capture Claim details

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

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

if (serviceFromMatch) jsonOutput.message.claim\_details["Service From Date"] = serviceFromMatch[1];

if (serviceThruMatch) jsonOutput.message.claim\_details["Service Thru Date"] = serviceThruMatch[1];

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

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

// Capture Member Info

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

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

if (dobMatch) {

const dob = dobMatch[1];

if (!/^\d{1,2}\/\d{1,2}\/\d{2,4}$/.test(dob)) {

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

}

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

}

// Check for Other Carrier Insurance (Do not have vs Does not have)

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

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

} else {

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

}

// Check if Member is calpers

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

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

} else {

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

}

// Remove null fields from JSON

function removeNulls(obj) {

if (typeof obj !== 'object' || obj === null) return obj;

const cleanObj = Array.isArray(obj) ? [] : {};

for (const key in obj) {

if (obj[key] !== null && obj[key] !== '') {

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

const nested = removeNulls(obj[key]);

if (Object.keys(nested).length !== 0) {

cleanObj[key] = nested;

}

} else {

cleanObj[key] = obj[key];

}

}

}

return cleanObj;

}

const cleanedJsonOutput = removeNulls(jsonOutput);

// Set the final output

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

}