A blue and black logo

Description automatically generated

Payment Collection Solution

Carl Newton - Solutions Architect

Cisco Systems

January 2024

Table of Contents

[Solution Goals 2](#_Toc156305099)

[Reduce Agent & Supervisor Effort 3](#_Toc156305100)

[Increase Customer Satisfaction Levels 3](#_Toc156305101)

[Create Data To Drive Outbound Campaigns 3](#_Toc156305102)

[JDS Event Injection 3](#_Toc156305103)

[Pre-Requisites 3](#_Toc156305104)

[Implementation 4](#_Toc156305105)

[Required Info 4](#_Toc156305106)

[Create OAuth Integration 4](#_Toc156305107)

[Install & Configure Postman Collection 5](#_Toc156305108)

[Create MockAPI DB 7](#_Toc156305109)

[Modify Agent Desktop Layout 9](#_Toc156305110)

[Create Flows In Webex 9](#_Toc156305111)

[Payment Due Workflow 9](#_Toc156305112)

[Payment Confirmed Workflow 15](#_Toc156305113)

[Test Flow 16](#_Toc156305114)

[Additional Features 17](#_Toc156305115)

[Quick SMS Message 17](#_Toc156305116)

[Pay By Link 17](#_Toc156305117)

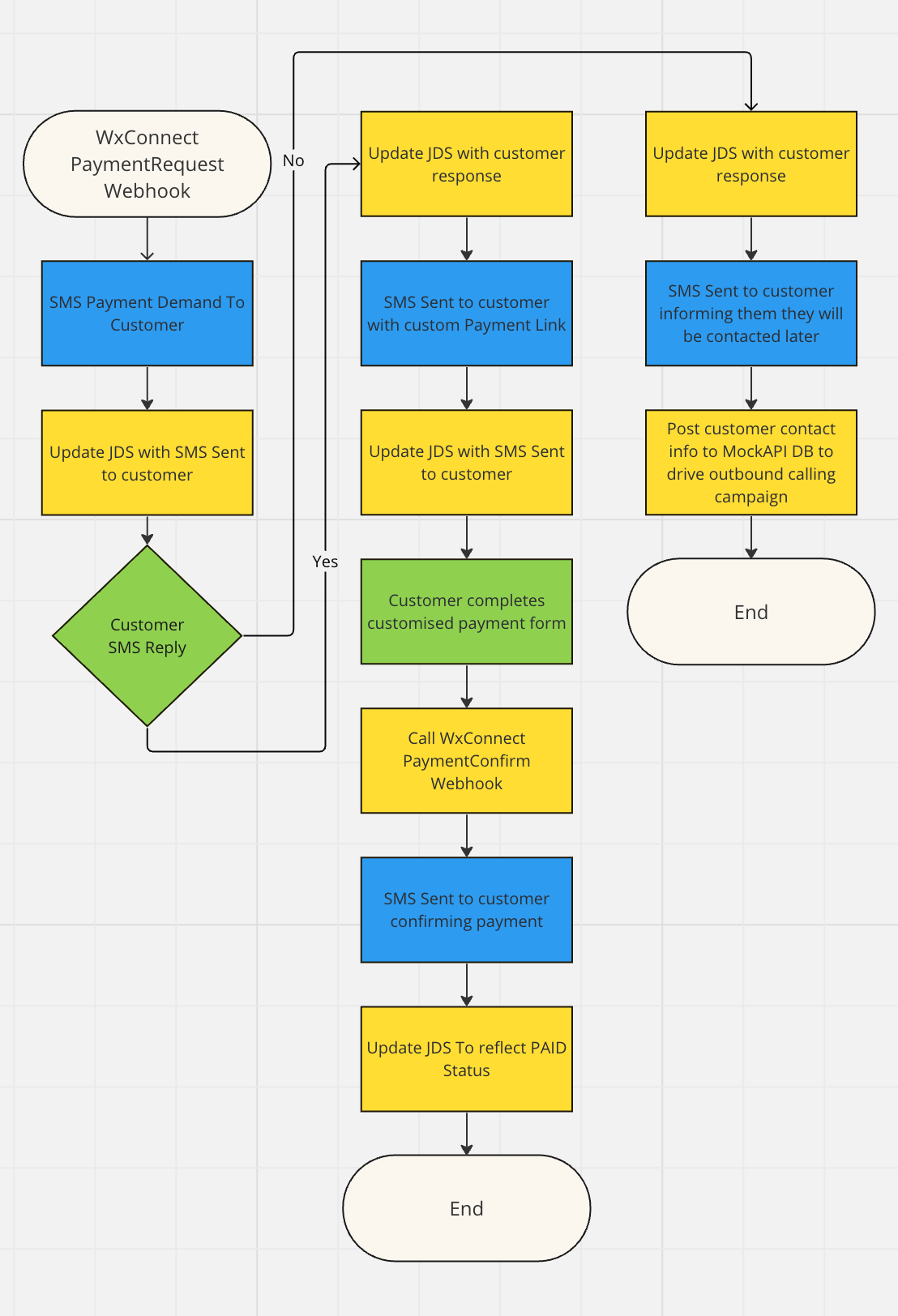
# Solution Goals

This document outlines the solution for a payment collection flow within Webex. Some customers currently use an outbound dialler to make outbound calls to clients and make payment demands. The efficiency of these solutions can be improved due to low call answer rates by the consumer who needs to pay, and high effort requirements for the Agents and supervisors to set up and execute the campaigns.

This solution has three aims:

* Reduce Agent & Supervisor effort by completely, or partially automating some of the collections
* Increase customer satisfaction levels by offering the ability to pay at a convenient time for them
* Create data about which customers have not paid to drive later outbound calling campaigns for agents to execute
* Inject information into the JDS history stream to assist agents in better handling customer interactions

The flow is described in the below flow chart



## Reduce Agent & Supervisor Effort

This is achieved by kicking off a Webex connect flow that can be invoked by a webhook. This flow will request payment from the customer by SMS, giving them the opportunity to pay with no interaction from any agent.

## Increase Customer Satisfaction Levels

Many people today do not answer calls from numbers they do not recognise. When they do answer, it may not be a good time for them to complete a payment. Being asked to complete a payment while on the phone to an agent at a bad time can greatly increase stress levels in what may already a stressful situation for the consumer. Ignoring those calls and later receiving final demands elevates stress even further.

Using SMS to drive payments allows people to pay when it is convenient for them. They are made aware of the demand in a non time sensitive manner, and can choose to pay at a more convenient time. This lowers consumer stress levels.

## Create Data To Drive Outbound Campaigns

If the consumer chooses not to pay at this time, the flow will add them to a list with the relevant details. This can then be used to drive a subsequent outbound calling campaign, or to kick off a number of other processes depending on the tools in use by the customer. Many integrations can be created to adjust this behaviour, but creating a list is a good way to show the customer the concept.

## JDS Event Injection

These flows also utilise the injection of JDS events into the history stream for a given user. This results in the agent being able to see payment demands being sent to a user, what their responses were, and any payments that have been made directly within the agent desktop. This could decrease AHT’s and increase satisfaction for the consumer, giving them the sense that the contact centre agent really knows all about them.

# Pre-Requisites

The following pre-requisites should be in place before attempting configuration

* An SMS asset in your Webex connect tenant capable of sending and receiving messages. [Details on Customer Engagement Portal](https://cisco.sharepoint.com/sites/GlobalCollaborationSalesEngineering/SitePages/Customer-Engagement-Field-Newsletter.aspx)
* [Postman](https://www.postman.com/downloads/) set up to initiate the flow
* Understanding of OAuth for Webex API executions
* Mockapi.io account set up to be later configured to receive contacts for the subsequent outbound calling campaign
* JDS Enabled WxCC Tenant (US Available, EU Roadmapped mid 2024)
* JDS capable WxCC Agent Desktop Layout that supports inbound/outbound calling
* [glitch.com](https://glitch.com/) account

# Implementation

## Required Info

This solution requires a few variables to be populated across different systems. The table below can be used to store this info. Note the names of the variable may deviate slightly across systems, so multiple names are included to assist identification. **DO NOT SHARE ANY OF THE FOLLOWING INFORMATION WITH ANYONE, ENSURE IT IS STORED SECURELY.**

|  |  |  |
| --- | --- | --- |
| Variable Name(s) | Variable Value | Relevant Section |
| Client ID  Client\_id.JDS | C12eb15a284c4ebc336049c41f263b105235942f32e4ec236ceaf5b5300b18c33 | From [Create Oauth Section](#_Create_OAuth_Integration) |
| Client Secret  Client\_sectet.JDS | 756174ef119c93e50adfca9c1f23c9f6919cb1a84f97944b866562d5ec12ebfd | From [Create Oauth Section](#_Create_OAuth_Integration) |
| OAuth Auth URL  OauthURL | https://webexapis.com/v1/authorize?client\_id=C12eb15a284c4ebc336049c41f263b105235942f32e4ec236ceaf5b5300b18c33&response\_type=code&redirect\_uri=https%3A%2F%2Fadmin.webex.com%2F&scope=cjp%3Auser%20spark%3Apeople\_read%20cjp%3Aconfig\_write%20cjp%3Aconfig%20cjp%3Aconfig\_read%20cjds%3Aadmin\_org\_read%20cjds%3Aadmin\_org\_write&state=set\_state\_here | From [Create Oauth Section](#_Create_OAuth_Integration) |
| Redirect URI | https://admin.webex.com/ | From [Create Oauth Section](#_Create_OAuth_Integration) |
| Project ID  Workspaceid  JDSProjectID  <<PROJECTID>> | 64da3f9d21a4034e04bc54c8 | From [Install Postman Section](#_Install_Postman_Collection)  From [Desktop Layout](#_Journey_Data_Services) |
| Org ID  Orgid.JDS  <<ORGID>> | 77dbece8-9e08-4e32-acd2-2f1a89c50d9a | From [Install Postman Section](#_Install_Postman_Collection)  From [Desktop Layout](#_Journey_Data_Services) |
| MockAPIID | 659ee28747ae28b0bd36e631 | [From MockAPI Section](#_Create_MockAPI_DB) |
| PaymentDue Webhook URL | https://hooks.us.webexconnect.io/events/OGWL6YI508 | [From Due Flow Configuration](#_Payment_Due_Workflow) |
| GlitchLink | swift-eight-cat.glitch.me | [From Due Flow Configuration](#_Payment_Due_Workflow) |
| Webhook.site URL  ReqCatcherURL | https://webhook.site/ec716c2e-562f-4669-8f44-b34ecc9c6657 | [From Due Flow Configuration](#_Payment_Due_Workflow) |
| PaymentConfirmed Webhook URL | https://hooks.us.webexconnect.io/events/85P18MBEYD | [From Confirmed Flow Configuration](#_Payment_Confirmed_Workflow) |

## Create OAuth Integration

An OAuth token is required to perform API operations for JDS event injection and other functionalities.

Important Note: OAuth tokens created may only last for a single day. Be sure to re-populate your token in your flows right before your customer demo to avoid failures. This will be described in a later portion of this document. Alternatively, set up a [token store](https://github.com/WebexSamples/webex-contact-center-api-samples/tree/main/token-management-samples/token-service-sample) using the [tutorial video](https://app.vidcast.io/share/ed971770-49bb-47e5-96d0-7c920074fd53) to automatically manage the tokens from flows.

1. Go to <https://developer.webex-cx.com/> and log in using an administrator account.
2. Click “My Webex Apps” from the sub menu in the top right, then select “Create a new app”  
   A screenshot of a computer

   Description automatically generated
3. Configure your integration as follows.

|  |  |
| --- | --- |
| Field | Value |
| Integration Name | PaymentDemand |
| Description | Integration to support SMS payment demand from Connect |
| Redirect URI | https://admin.webex.com/ |
| Scopes | Select All |

1. You will be presented with a screen of information. Go to the [required information](#_Required_Info) table and fill out the relevant fields with the info displayed on the screen.

## Install & Configure Postman Collection

The postman collection has several functionalities. Download it from the [Git Repo](https://github.com/canewton-ccep/PaymentCollections). This collection uses JDS API samples by Arunabh and Barry McLellan located [here](https://github.com/WebexSamples/webex-contact-center-api-samples/blob/main/customer-journey-samples/cjds-postman-example/JDS%20CiscoLive.postman_collection.json)

1. Open Postman, choose File -> Import
2. Load the file “CCEP Payment Collection.postman\_collection.json”
3. Log into control hub. Select the “Customer Journey Data” Tab, and copy your project ID. Populate it, and your Org ID in the [Required Info Table](#_Required_Info).  
   A screenshot of a computer

   Description automatically generated
4. Go back to Postman. Select the collection you just imported, and go to the variables screen. Populate all of the fields using info from the [Required Info Table](#_Required_Info). except for MockAPIID, baseURL & identity, these will be used later. **SAVE THE COLLECTION**A screenshot of a computer

   Description automatically generated
5. Click on the Authorization tab. Scroll down to the “Configure new token” area. The fields should be pre-populated to reference the variables you just set up.   
   Scroll to the bottom and select “Get New Access Token”. You will be asked to authenticate, and then accept the integration.  
   A screenshot of a computer

   Description automatically generatedA screenshot of a computer

   Description automatically generated
6. On success, You will be presented with the token details. Select “Use this token”. Feel free to close this screen for now. If the token fails, check the postman log to see why and re-try. Clearing cookies before choosing “Get new access token” can help.
7. Open the “JDS” Section of the collection, and open the “Get History Stream” request. Now just click “Send” and you should receive a 200OK and get some kind of data. Do not proceed until you have a 200OK.

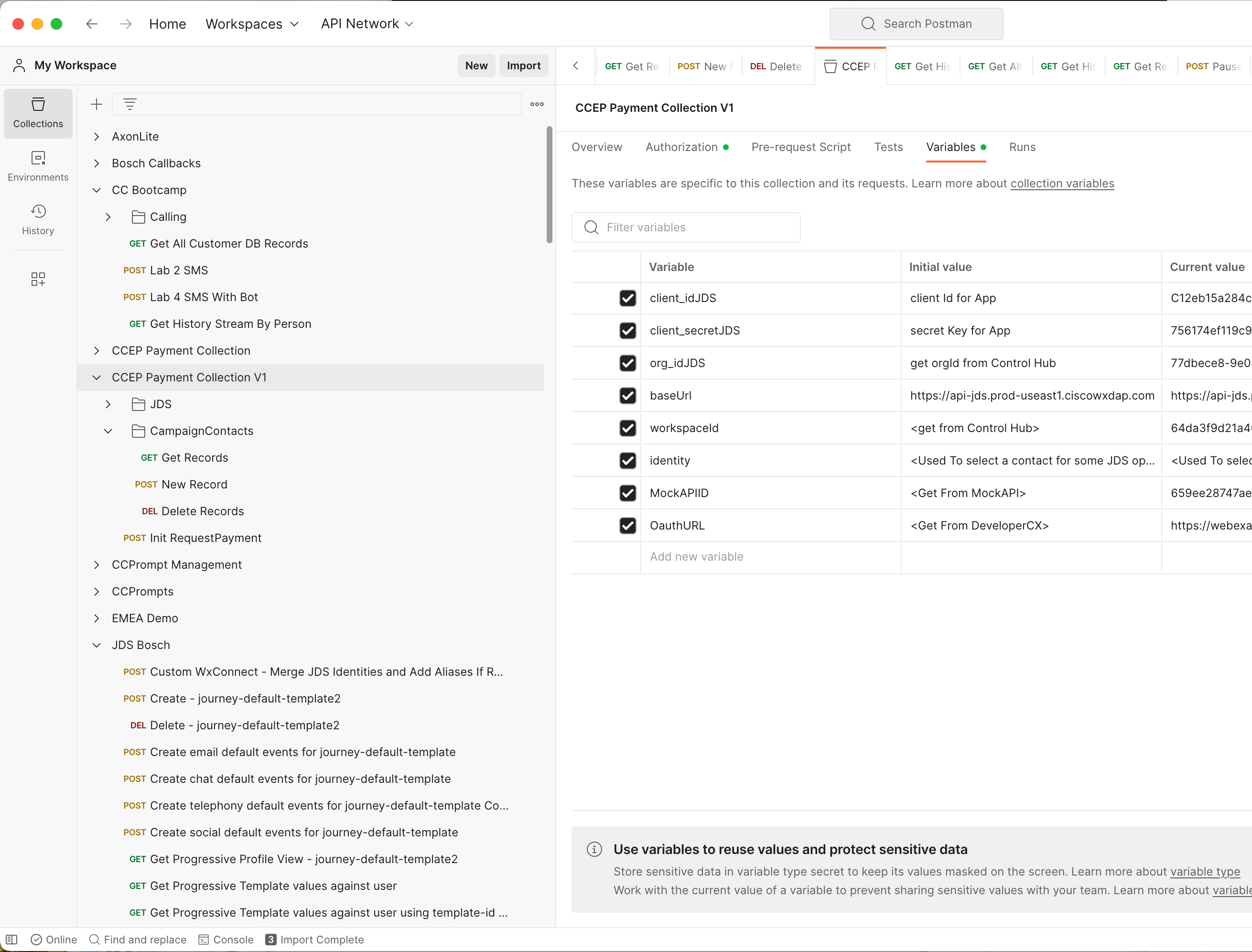
A screenshot of a computer

Description automatically generated

## Create MockAPI DB

You will need to create a MockAPI account to act as the “Receiver” for the users that chose not to pay the SMS demand. It will contain the details of all of the consumers that need to be added to any outbound calling campaign the customer chooses to run.

1. Go to <https://mockapi.io/> and create an account
2. Create a new project called WXCC
3. Open your new project and copy out the code part of the API endpoint. Populate this in the [Required Info Table](#_Required_Info), and in the postman variables screen we populated earlier under MockAPIID  
   A screenshot of a computer

   Description automatically generated  
   
4. Create a New Resource called “collections”  
   Note the name is case sensitive and the solution will fail to function properly if it is named anything else. Your endpoint URL must be similar to <https://your_endpoint_code.mockapi.io/collections> Give the collection schema settings as follows  
     
   A screenshot of a chat

   Description automatically generated
5. Click somewhere in grey bar with a 0 to create some records using faker.js, then click the Data button to see the created data.  
     
   A screenshot of a computer

   Description automatically generated
6. Now run the “Get Records” request from postman. Ensure you see data.  
     
   Feel free to test the add record and delete record requests from postman as well. The delete record request works by referencing the ID of the record to delete in the request URL, so you will need to populate it:  
     
   https://{{MockAPIID}}.mockapi.io/collections/**1**  
     
   Will delete record id 1 from the following output:

{

"FirstName": "Manuel",

"LastName": "Tremblay",

"PhoneNo": "832-409-8902 x4241",

"id": "1"

},

{

"FirstName": "Delphia",

"LastName": "Borer",

"PhoneNo": "294-438-6214 x7685",

"id": "2"

},

{

"FirstName": "Estella",

"LastName": "Howell",

"PhoneNo": "247.249.1126 x4669",

"id": "3"

},

## Modify Agent Desktop Layout

Next, we will modify the provided agent desktop layout to work with your instance of contact centre. There are a few changes required to get each feature working. Download the file “CCEP\_Payment\_Collection\_Layout.json” from the Git repository and open in your favourite code editor. Visual Studio Code is free and recommended.

The JDS Widget is important in this use case, as it shows the history of payments and text messages made by the user, in addition to calls.

Find the following section around line 61 of the layout file and replace <<PROJECTID>> with the project ID and <<ORGID>> with the info from the [required info table](#_Required_Info). Do the same again at around line 321 (The info appears in two places, once for the “in call” panel area widget, and again for the out of call, nav bar area widget)

"comp": "customer-journey-widget",

"script": "https://cjaas.cisco.com/widgets/customer-journey-9.0.0.js",

"attributes": {

"base-url": "https://api-jds.prod-useast1.ciscowxdap.com",

"logs-on": "true",

"project-id": "<<PROJECTID>>",

"live-stream": "true",

"icon-data-path": "https://amazon-connect-104c098076f3.s3.eu-central-1.amazonaws.com/WxCC-JDS/icons\_v3.json"

},

"properties": {

"interactionData": "$STORE.agentContact.taskSelected",

"bearerToken": "$STORE.auth.accessToken",

"organizationId": "<<ORGID>>"

Now upload your desktop layout to WxCC and assign it to an agent you want to test with.

## Create Flows In Webex

The next step is to create the flows within Webex. There are two flows to implement.

### Payment Due Workflow

1. Log into Connect and create a new service called “PaymentCollection”
2. Go to Flows and create a flow. Name it PaymentDue and upload the CCEP\_PaymentDue.workflow file
3. A configuration page will appear titled “Configure Webhook”. Choose the box for “Create new event”, name it “CCEP\_Paymentdue” and populate the sample input with the below JSON. Click the Parse button, copy the webhook URL into the [required info table.](#_Required_Info) And then click save.  
   {

"MobNum": "+447571",

"FirstName": "Carl",

"LastName": "Newton",

"PaymentDue": "€198",

"DueDate": "1/1/2024",

"RefNum": "1234"

}

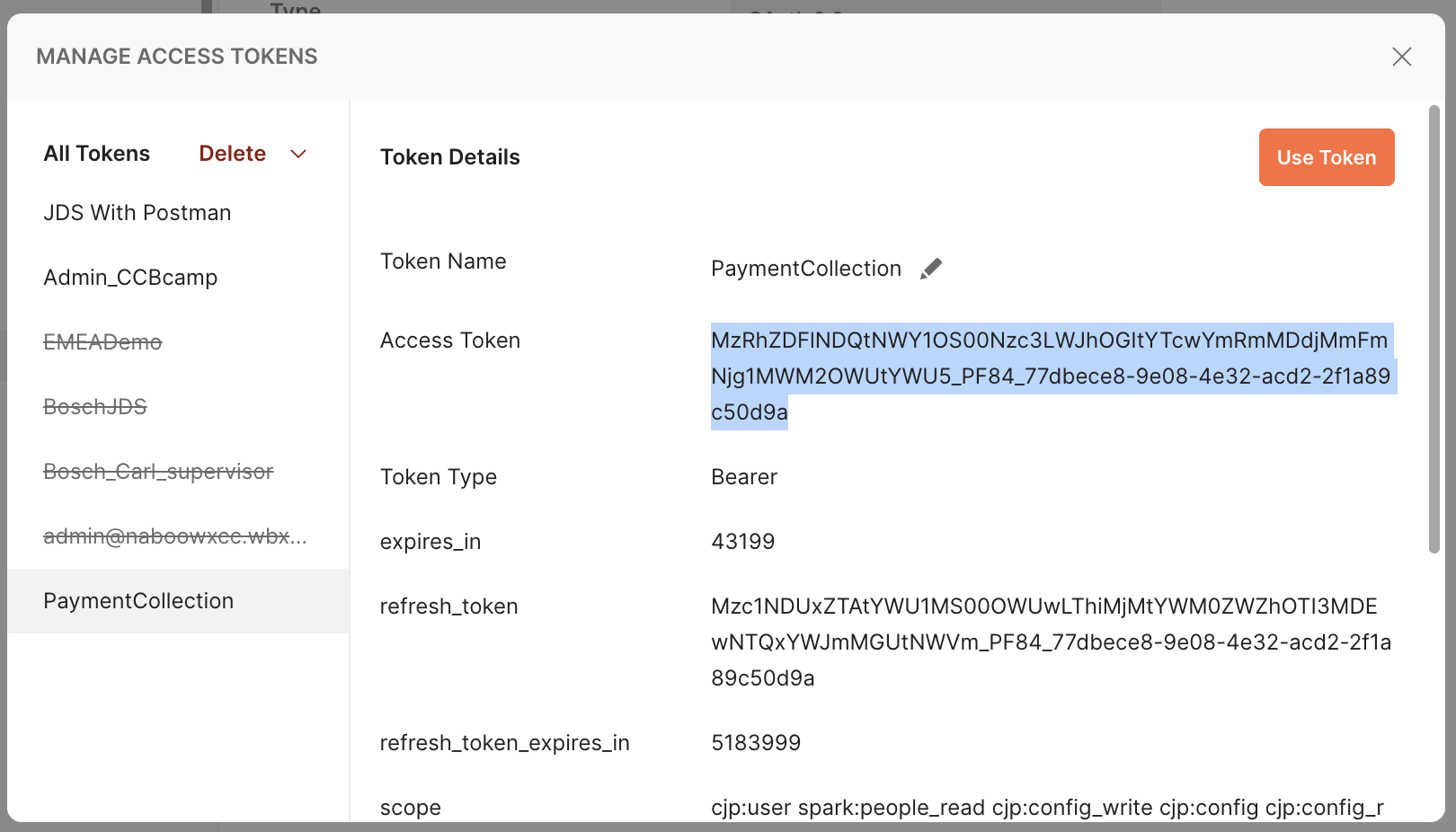
A screenshot of a computer

Description automatically generated

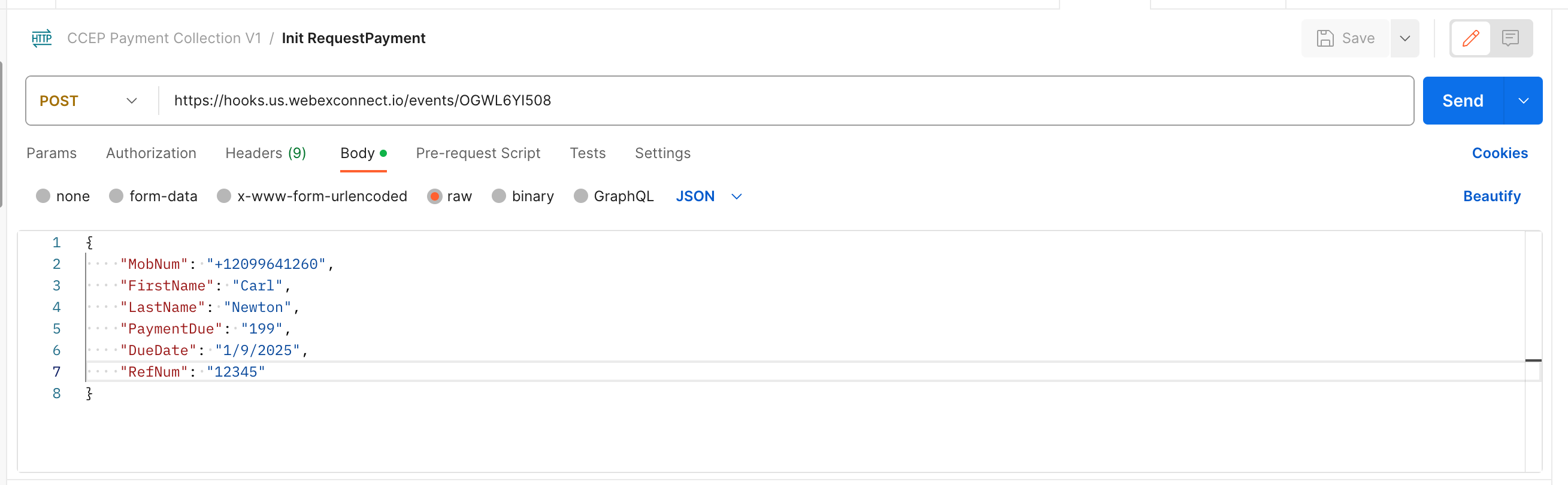
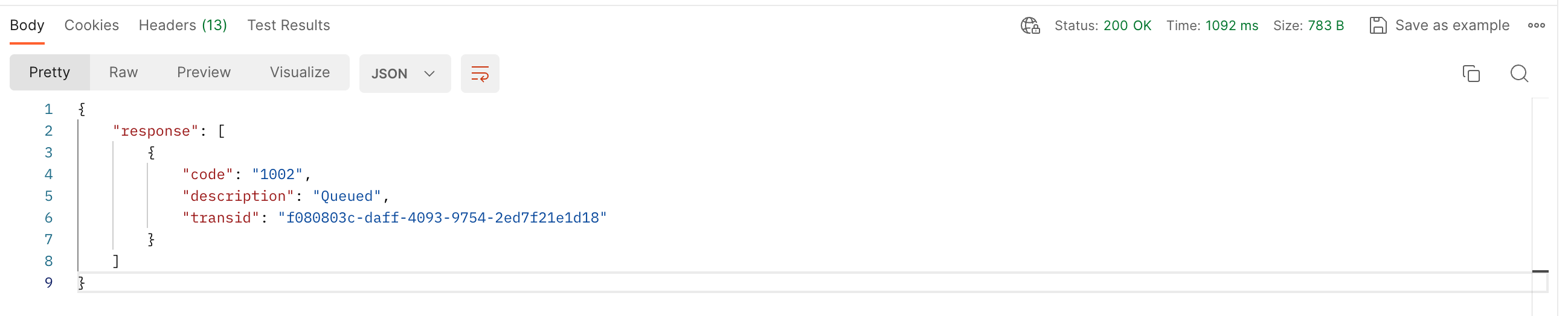
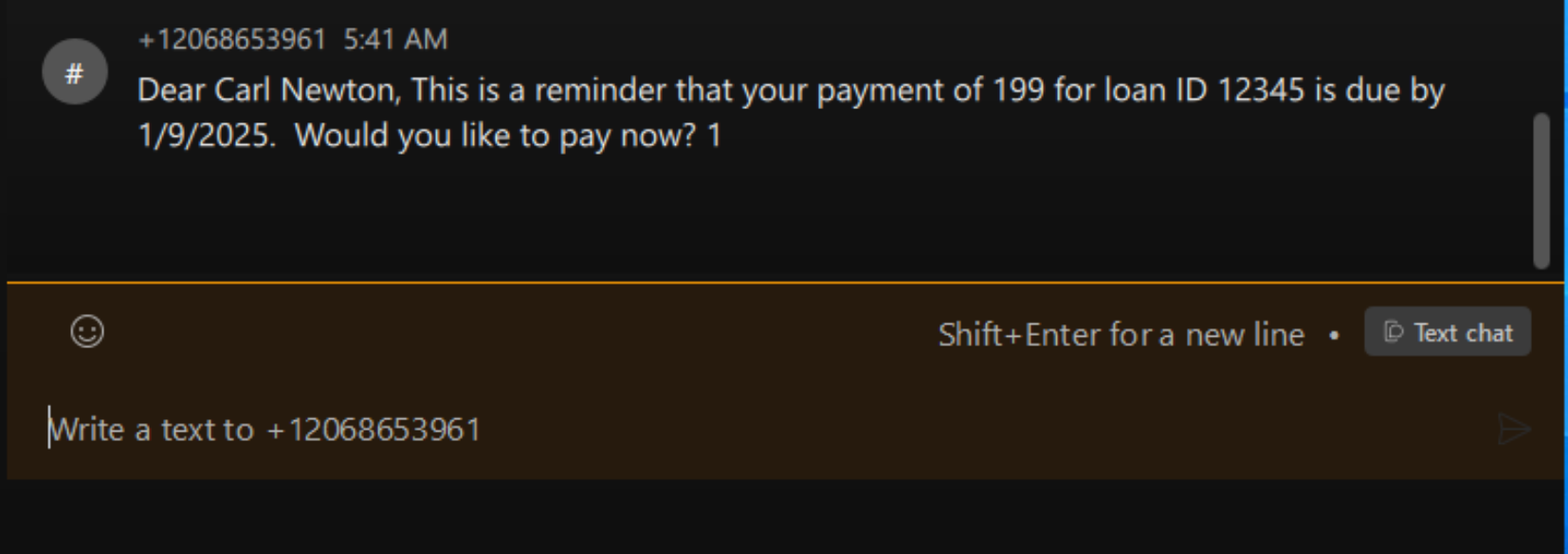
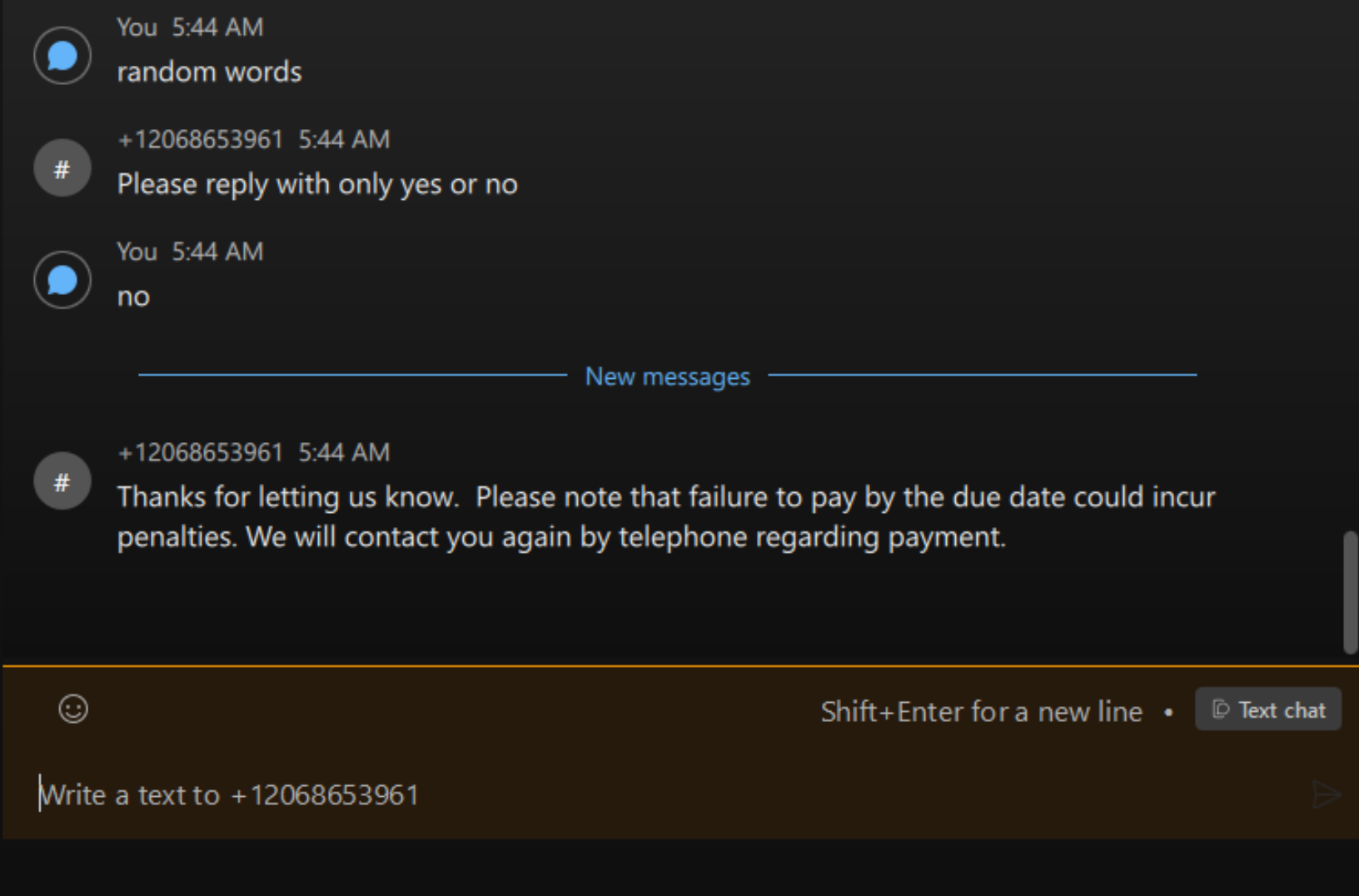
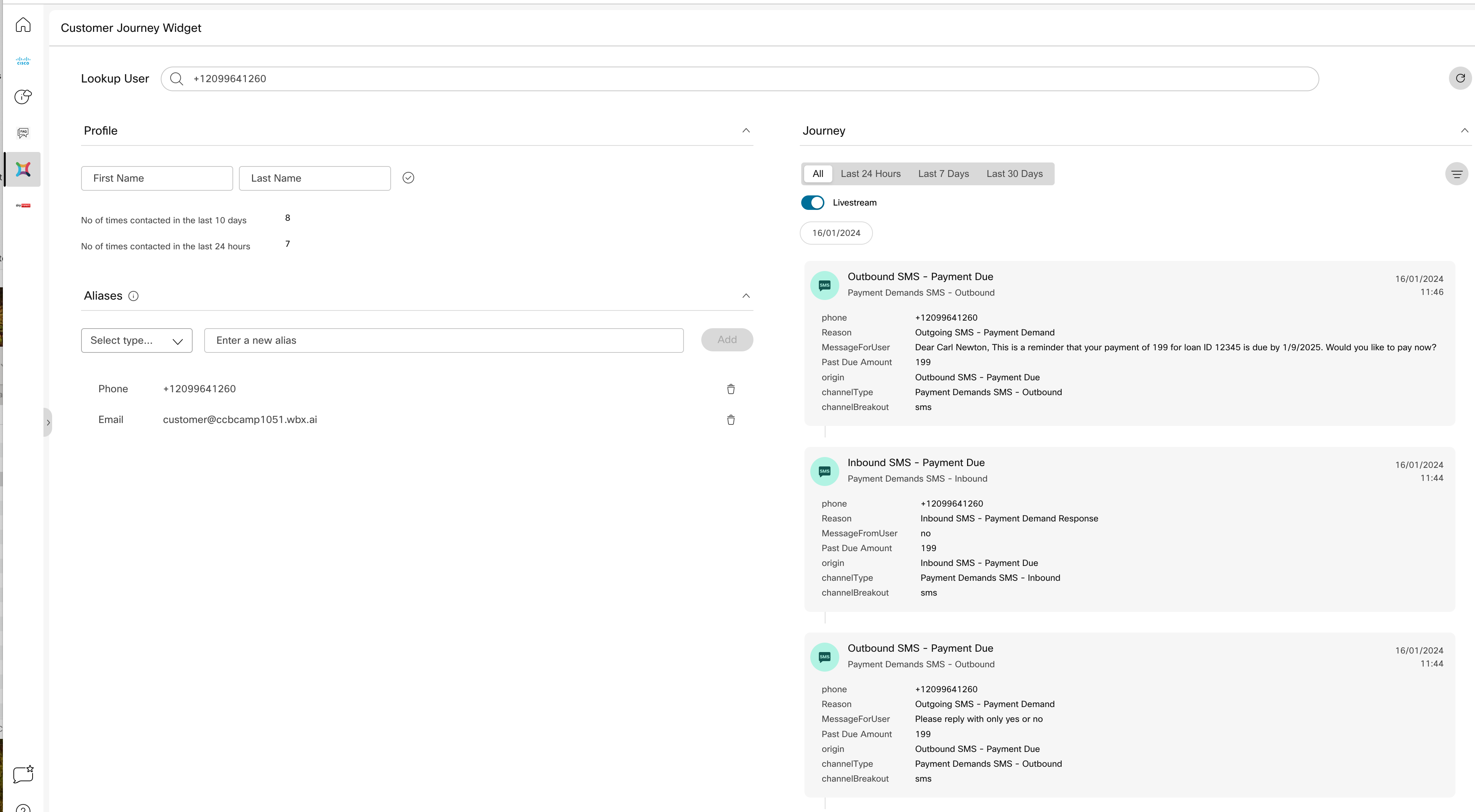
1. Now double click all of the SMS nodes (There are four) and select the desired “From Number” of the SMS asset within Connect. Choose save.  
     
   A screenshot of a computer

   Description automatically generated
2. Now double click the Receive node, and choose the same number from the drop down list. Set the keyword as \*  
     
   A screenshot of a computer

   Description automatically generated
3. Next, we will set up some debugging so we can see the flow working. Open a new tab to https://webhook.site/ and copy out the Unique URL. Populate it in the [required info table.](#_Required_Info) You can create your own nodes within the flows we will set up to send debugging information to webhook.site in order to troubleshoot the flows if they are not working correctly.
4. There is a glitch site that has some back end code we need to use here. We will configure fully later, for now, open <https://glitch.com/edit/#!/swift-eight-cat> and click the remix button. You will receive a new URL. Copy \*Only\* the site identifier into the [required info table](#_Required_Info) (E.g swift-eight-cat.glitch.me)
5. Now click the settings cog in the top right of the WxConnect flow builder. Go to the “Custom variables” tab. Populate all of the fields except for JDSToken from the [required info table.](#_Required_Info) and leave the page open.
6. Go to postman, click the “CCEP Payment Collection” collection, and then the Authorization tab. In the “Current Token” area, notice that the token expiry time is listed. If it is before your demo will happen, make sure to refresh the token at a time that gives you a valid token for the demo. Next, click the drop down, and then manage tokens.  
     
   A screenshot of a web page

   Description automatically generated
7. Choose your “payment Collection” token, copy out the access token, and paste it into the “JDSToken” custom variable field within Webex Connect. Ensure you do not have any whitespace or carriage returns as part of the pasted string. Now save the custom variables in Connect, save the flow, and make live.  
     
   

#### Test Flow

1. Open postman, and select the “Init RequestPayment” request. Ensure the type is “POST” and the address is your webhook from the flow we imported earlier. Go to the “Body” Tab, and populate the fields with your relevant info. Make sure to include a number that can receive SMS. In the below example, a “business SMS” enabled sandbox Webex account is used. When presenting to customers, it is more impactful to share a mobile phone screen into a meeting; but be mindful of international SMS charges. Click save, and send the request. See below for a screenshot of the request, and the 200OK response from WxConnect.  
     
   
2. Monitor your endpoint, you should receive a text corresponding to the information you provided in the body of the postman request.  
   
3. Reply with some random words. You should receive a text asking you to reply yes or no. Next, reply with the word “No”  
   
4. Now check your mockAPI DB. You should see an entry for this user in the DB. This is what you would present to the customer as a list to drive an outbound calling campaign in the event that a customer chose not to pay the SMS based demand.  
   
5. Now, check your agent desktop, choose the JDS widget, and enter the number of the consumer. In this example, +12099641260. You should see the full stream of inbound and outbound SMS between connect and the user. If you do not, re-populate the JDS token in the custom variables of the Webex Connect flow using a fresh token retrieved from Postman and re-test. If it still does not work, use HTTP Post to webhook.site to debug. In this case, any time an agent speaks to a customer, the JDS widget would automatically load this full history.  
   

### Payment Confirmed Workflow

1. Open your “Payment Collection” service in WxConnect
2. Go to Flows and create a flow. Name it PaymentConfirmed and upload the CCEP\_PaymentConfirmed.workflow file
3. A configuration page will appear titled “Configure Webhook”. Choose the box for “Create new event”, name it “CCEP\_PaymentConfirmed” and populate the sample input with the below JSON. Click the Parse button, copy the webhook URL into the [required info table](#_Required_Info). And then click save.

{

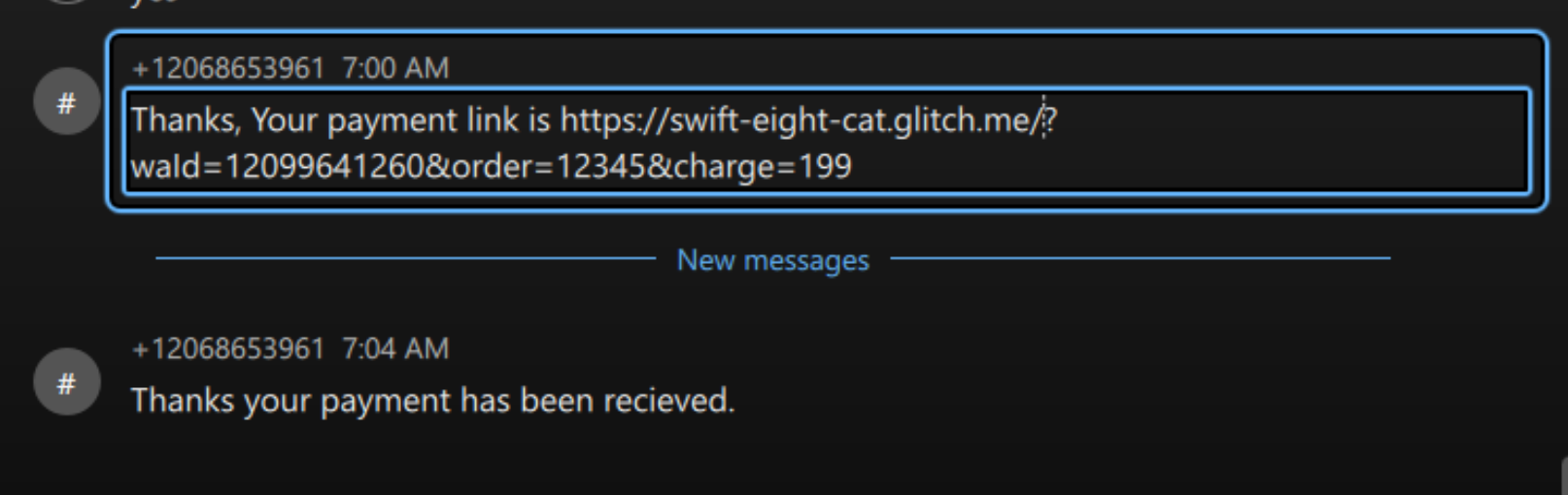
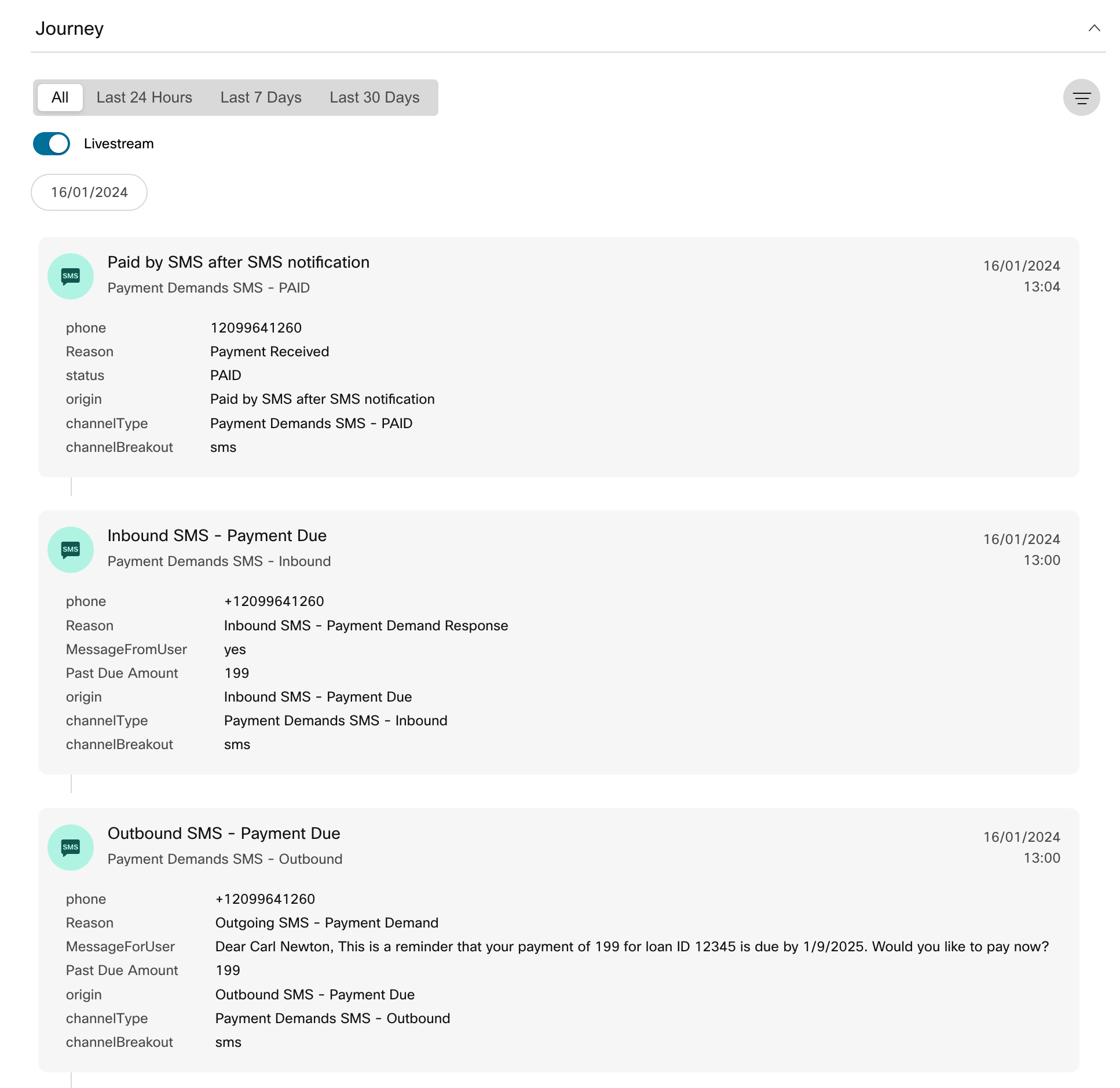
"waId": "12099641260"

}

1. Open the SMS Node and change the “From Number” to your SMS Number
2. Open the custom variables and populate all the fields from the [required info table](#_Required_Info)
3. Save and publish the flow
4. Open your glitch site and select the script.js file. Around line 33, you will see a statement like below. Change the address to the webhook for the paymentconfirmed flow.  
   url: 'https://hooks.us.webexconnect.io/events/85P18MBEYD',

### Test Flow

1. We are now ready to test this part of the flow. Start your postman “Init Paymentrequest” and reply yes to the text message
2. You will receive a payment link. Open this in a browser. Fill out any details for the card numbers etc, this is a mock payment system. Submit the details and observe that the status changes to paid  
   A screenshot of a payment

   Description automatically generated
3. Your consumer should receive a text message confirming payment.  
   
4. Observe in the agent desktop that the confirmed payment is showing in the JDS history  
   

# Additional Features

THIS SECTION REMAINS TO BE FINISHED

## Quick SMS Message

## Pay By Link

This widget has been adapted from the original author Adam Cohen’s acohen67@cisco.com implementation. allows an agent to send a connected customer a payment link in real time, and view the status of that payment as the customer pays. It calls two webhooks during operation, one to send a payment link, and another periodically to check if the payment has been made. Once the consumer completes the payment form, the agent can see it within the desktop.

A screenshot of a payment

Description automatically generated A screenshot of a payment card

Description automatically generated A screenshot of a payment card

Description automatically generated

Find the following section around line 100 of the layout:

"comp": "pay-widget",

"script": "https://<<GLITCHID>>.glitch.me/payment.js",

"attributes":{

"btncolour" : "#0045A0"

},

"properties": {

"customerPrefix": "gen",

"currentTask": "$STORE.agentContact.taskSelected",

"triggerPaymentWebhookURL" : "<<TRIGGERHOOK>>",

"checkPaymentSyncWebhookURL" : "<<CHECKHOOK>>",

"syncWebhookKey" : "<<SYNCKEY>>",

"pollInterval" : "5000",

"maxAttempts" : "10"