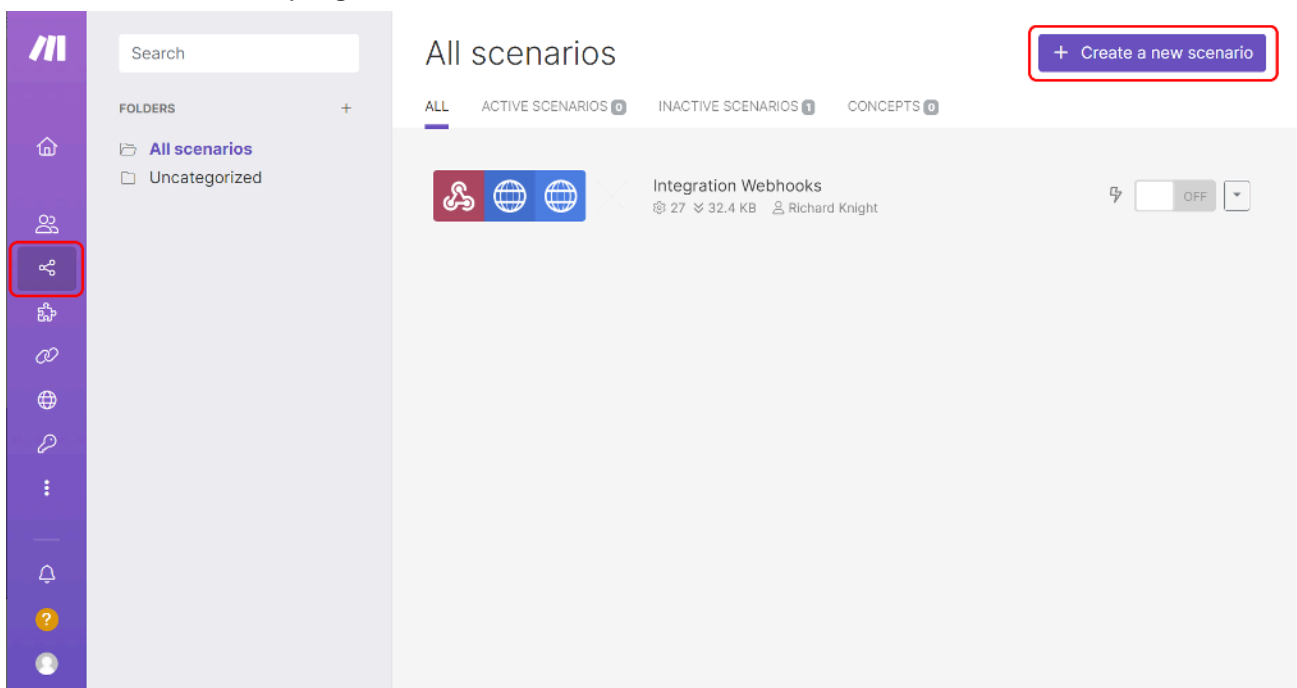


Using Webhooks and API to Manipulate Data in GW Apps

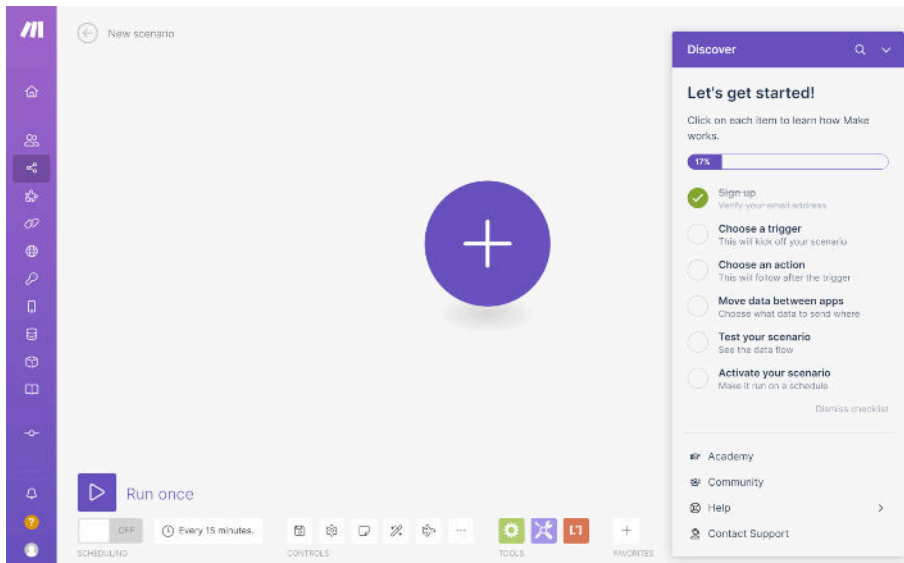
This document will describe an example of using a webhook from GW Apps and API calls back into GW Apps to process data in GW Apps. This is a very simple example, as we wanted to focus on the process of using GW Apps webhooks and API Calls with Make. Once you have followed these steps, you can expand this base to enable integrations with countless external systems or specific advanced functionality not currently available.

In this exercise, we will use Make to set the contents of a text area field to a specific value. We will then extend it to read a date from a date field, add 5 days to the date, format it properly and write that to the same text field. If you don't currently have an account at Make, you can create a free one. This will allow you to create 2 scenarios (set of steps that accomplish a specific task), one of which you could use for this example.

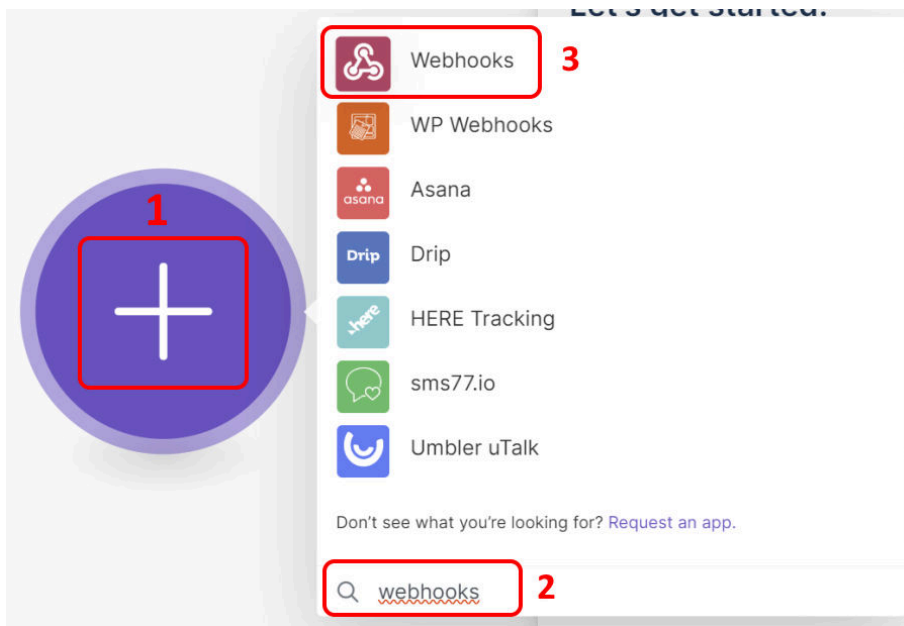
- A.** In **GW Apps**, either create a new application or find an existing 'test' application that you can modify. For this exercise, all you will need is a very simple app, that you have Designer rights to, with a form and that has a single Text or Text Area field whose value you can update. If you are using an existing app, either add a new form with a single Text or Text Area field or identify an existing form that has an editable Text or Text Area field.
- B.** Launch **Make**, click on Scenarios from the left-side navigation and then '+ Create a new scenario' from the top right of the Scenarios screen:



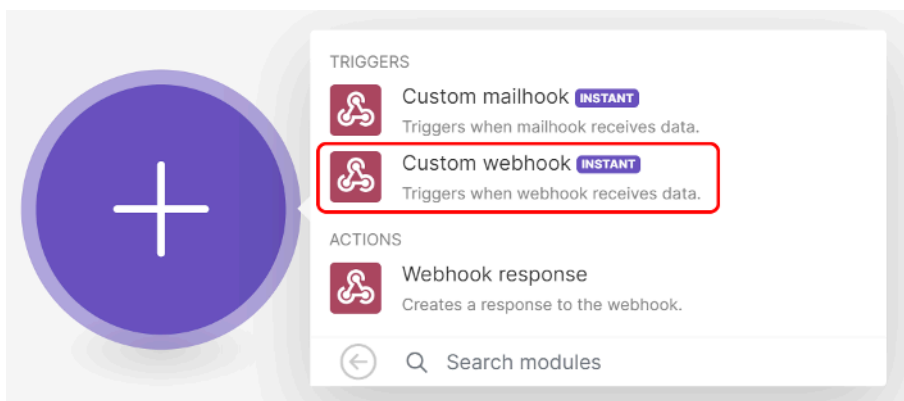
You will now see your new, empty scenario. Follow the steps below to configure it, and GW Apps, to make the webhook and API calls that will update your GW Apps test record.



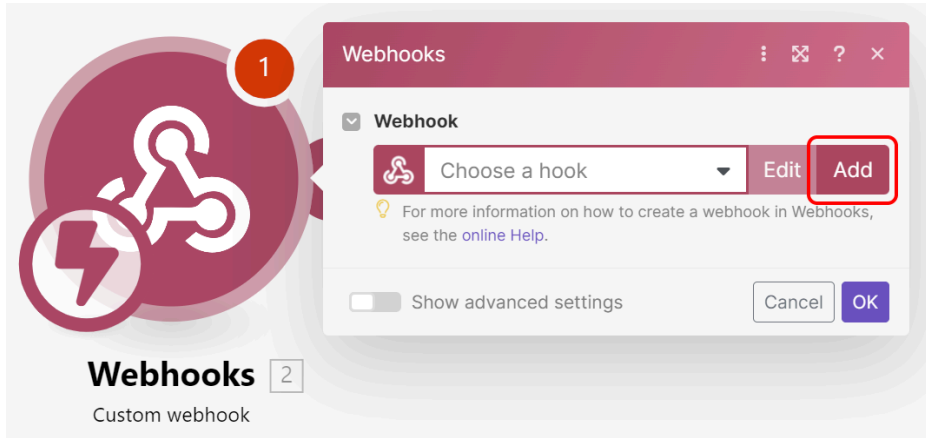
- C. Click on the '+' in the middle of the starting circle (1).
- D. Then type "webhooks" in the search bar at the bottom of the dialog that appears (2).
- E. Then click on the 'Webhooks' option in the list (3).



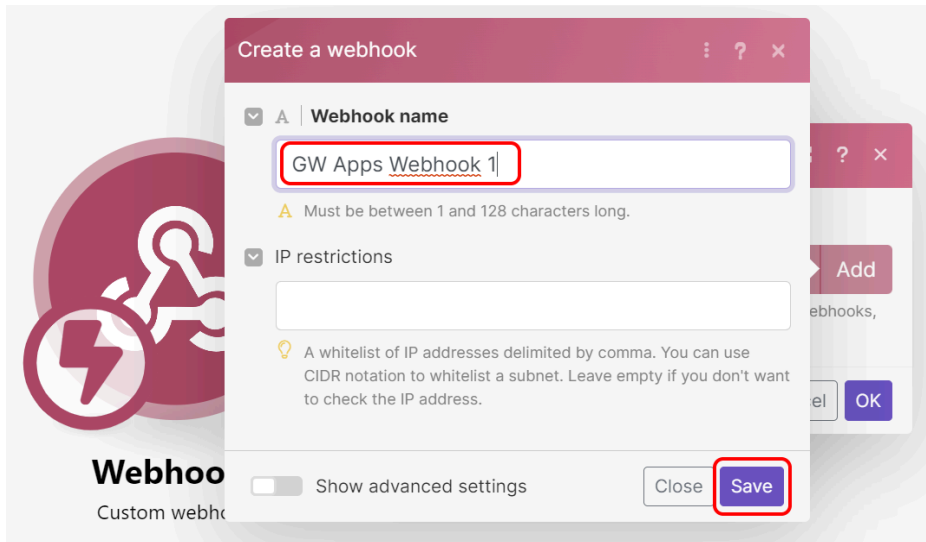
- F. Select 'Custom webhook'



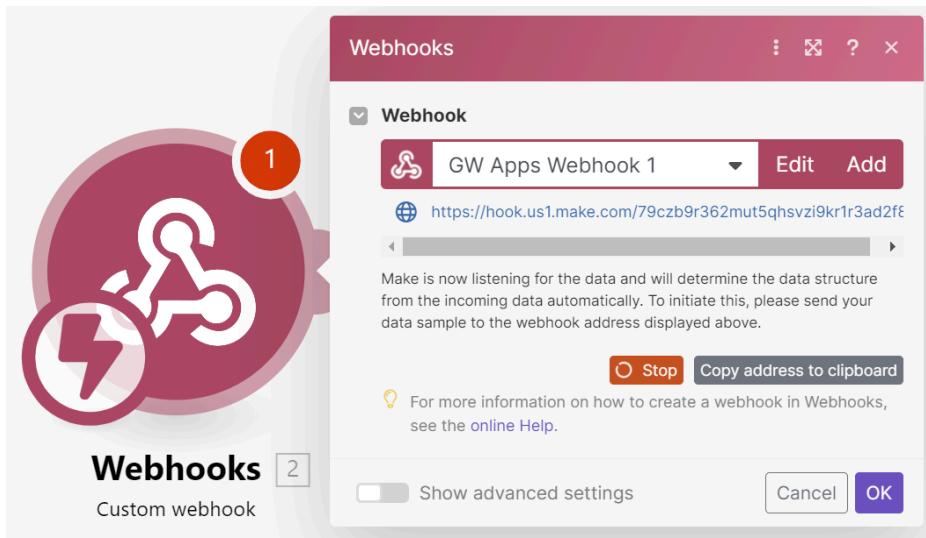
G. Click on 'Add' to add a webhook.



H. Type in the desired name for this webhook. The name has no functional effect, so just pick a name that makes sense in the context of this scenario. Then click 'Save'.



I. You will now see the webhook listening for incoming requests.



- J.** Open **GW Apps** and go to the Test App you wanted to integrate with Make. Open the Actions list (Edit App > Actions) and create a new action and set the following values:
- a. Action Type: Webhook
 - b. Action Name: Any suitable name
 - c. Select From: Select the form that has the data you want to manipulate.
 - d. Webhook URL: We need the value from Make. See next step for details.

New Action

Select Action Type * **a** HTTP Webhook

Action Name * **b** Make Webhook

Webhook: You can use Webhooks to post to any web service. This is a great way to integrate GW Apps with a variety of popular services such as Slack, Google Meet, Microsoft Teams, Zapier and many more.

Description

Method * POST Protocol * HTTPS

Webhook URL * **d**

Select Form * **c** Text Area

Cancel Create

- K.** Go back to **Make**, and copy the URL of the webhook you just created there:

Webhooks

Webhook

GW Apps Webhook 1 Edit Add

https://hook.us1.make.com/79czb9r362mut5qhsvzi9kr1r3ad2ff

Make is now listening for the data and will determine the data structure from the incoming data automatically. To initiate this, please send your data sample to the webhook address displayed above.

Stop Copy address to clipboard

For more information on how to create a webhook in Webhooks, see the [online Help](#).

Show advanced settings Cancel OK

Webhooks 2 Custom webhook 1

- L. Go back to **GW Apps**, and paste the webhook URL into the Webhook URL field in the Create Action Dialog, then click 'Create'.

New Action [Close]

Select Action Type * **HTTP Webhook** Action Name * **Make Webhook**

Webhook: You can use Webhooks to post to any web service. This is a great way to integrate GW Apps with a variety of popular services such as Slack, Google Meet, Microsoft Teams, Zapier and many more. 12 / 128

Description
Description

Method * **POST** Protocol * **HTTPS** **Webhook URL ***

Select Form *
Text Area

Cancel **Create**

- M. You will now see the created webhook action and you can complete it's configuration:

Webhook Configurations: Make Webhook

Name * **Make Webhook** Description

Request Details

Method * **POST** Protocol * **HTTPS** Webhook URL * **hook.us1.make.com/79czb9r362mut5qhsvzi9kr1r3ad2f8y**

Security

Authentication Type * **None**

Header Details

+ Add Header

Request Body

Request Body **Full Record** Request Content Type * **Application/Json**

Response Details

Accepted Status Codes **Range of Codes** Range From **200** Range To **300**

Request Timeout **10** secs Retries **Don't Retry**

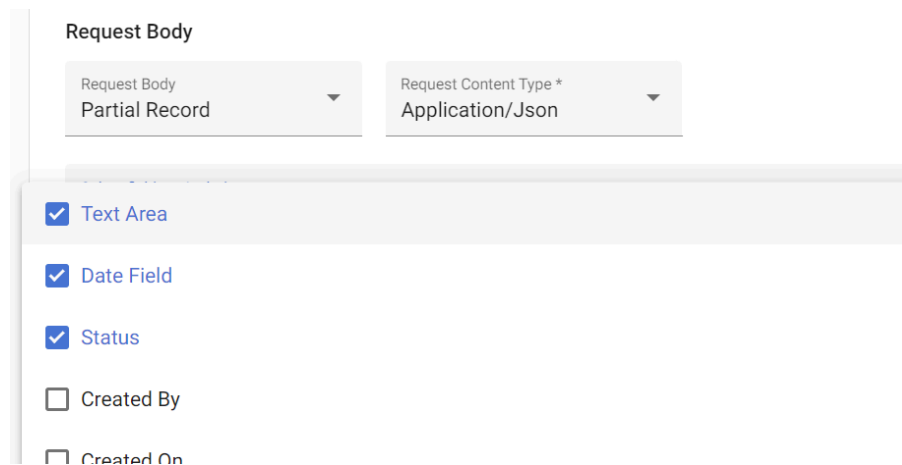
Cancel **Save**

The dialog has the following sections:

- Request Details: These settings are already completed.
- Security: Configure as required by the receiving webhook. Options include: None, Basic Auth, Bearer Token, and JWT. In this example, it can be left as 'None'.
- Header Details: Configure as required by the receiving webhook. You can add as many Key-value pairs as needed. In this example, we do not require any headers.
- Response Body: Defines what information will be sent with the webhook. Details of the options are discussed below.
- Response Details: These settings can usually be left as-is, unless you have a specific situation that would need them to change.

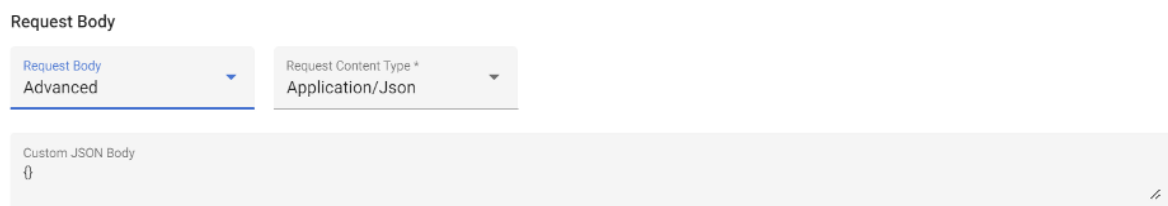
The Request Body section has three options: Full Record, Select Fields and Advanced:

- Full Record - All data related to the record will be sent as the webhook body.
- Partial Record - Selecting this option will display the 'Select fields to include' field which is a multi-select drop-down which will allow you to select the specific data items you wanted sent as the webhook body. See example below:



The screenshot shows the 'Request Body' section of a configuration dialog. At the top, there are two dropdown menus: 'Request Body' set to 'Partial Record' and 'Request Content Type *' set to 'Application/Json'. Below these is a multi-select dropdown menu titled 'Select fields to include'. The selected options are 'Text Area', 'Date Field', and 'Status'. Unselected options are 'Created By' and 'Created On'.

- Advanced - Selecting this option will display the 'Custom JSON Body' field. Enter the desired JSON body content. See example below:

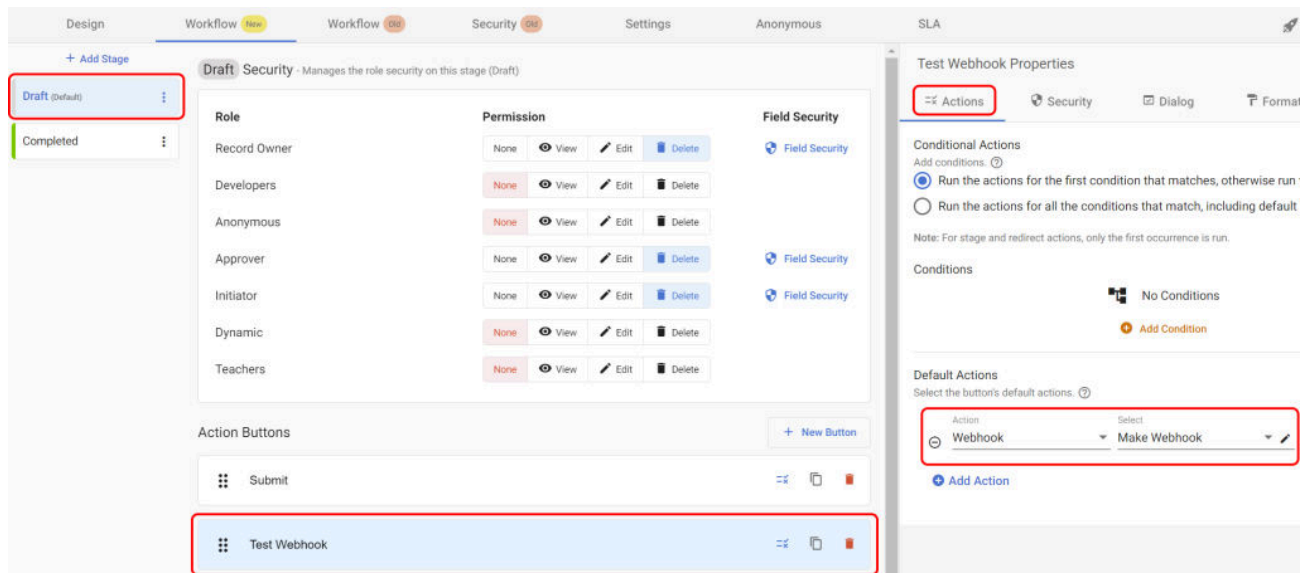


The screenshot shows the 'Request Body' section of a configuration dialog. At the top, there are two dropdown menus: 'Request Body' set to 'Advanced' and 'Request Content Type *' set to 'Application/Json'. Below these is a text area labeled 'Custom JSON Body' containing the characters '{}'. A small icon is visible in the bottom right corner of the text area.

Note: You have to use a valid json. Make sure that your properties are all wrapped as a string.

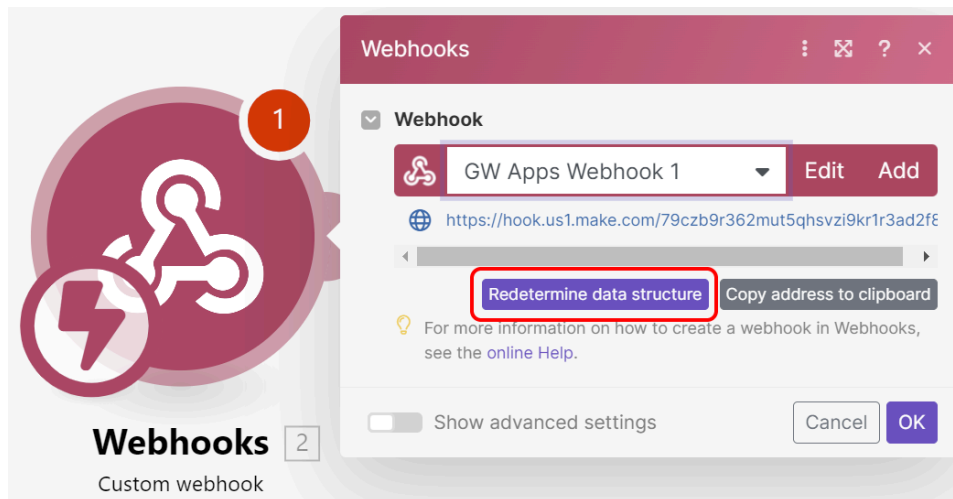
Select 'Partial Record' for the Request Body, and then select 'Status' and your Text or Text Area field. ('Text Area' is the name of the field in our example form.)

N. Now configure a workflow action button to trigger this webhook. (The 'Test Webhook' option in the actions list is not a suitable way to test this webhook as we need it to send data from a valid record as well.) The easiest way is just to create a temporary "Test Webhook" action button that can be clicked while viewing or editing a test record.

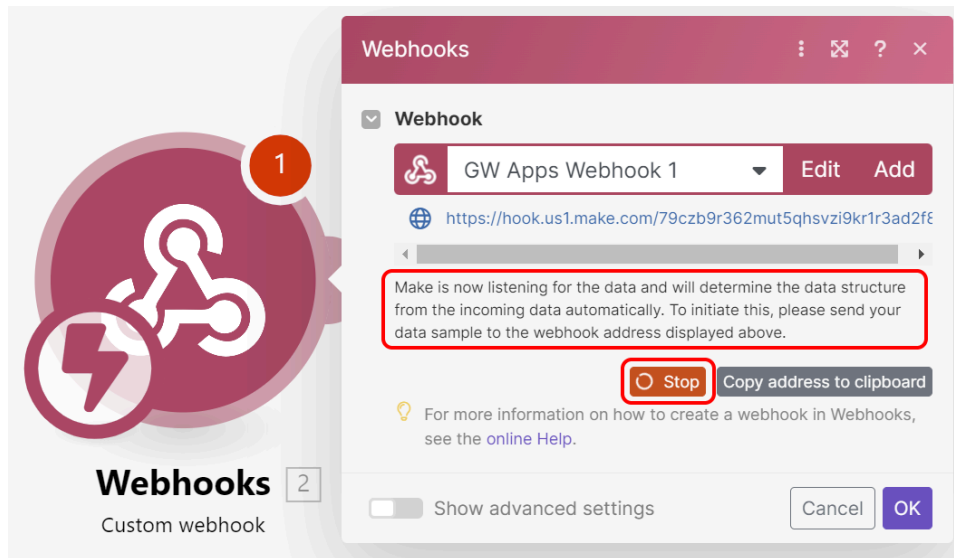


O. We need to trigger the webhook so that Make can receive a sample data body and parse it. This is a two step process. First we need to ensure the webhook in Make is listening, and then we need to trigger the webhook action in GW Apps to send the data.

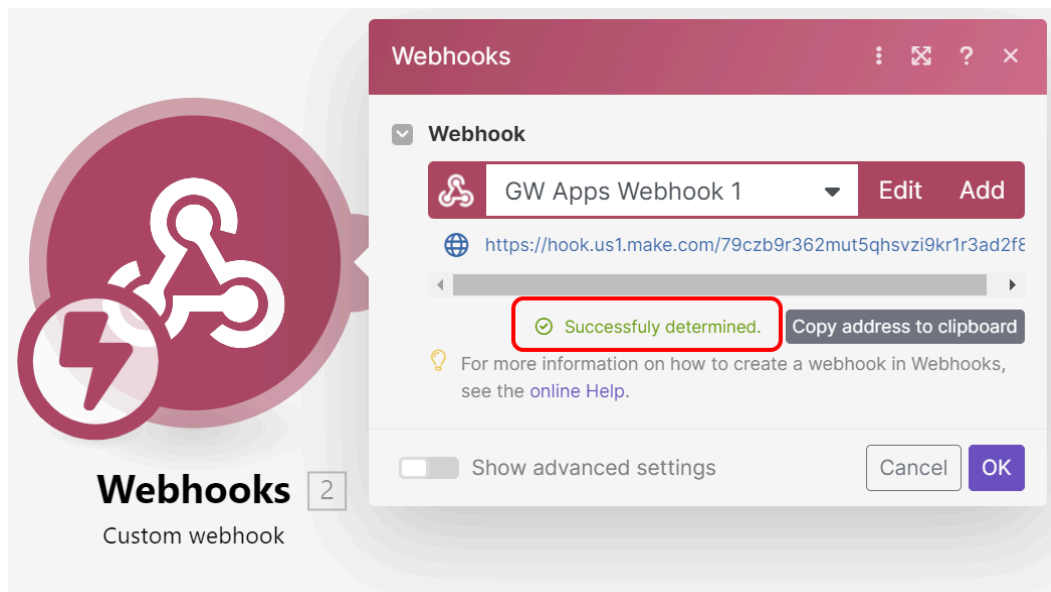
Go back to **Make**, and click on 'Redetermine data structure'



The dialog will now show a message letting you know it is now listening for the data, along with a 'Stop' button in case you wanted to stop the webhook listening, for example if you realized you weren't quite ready to test after all:

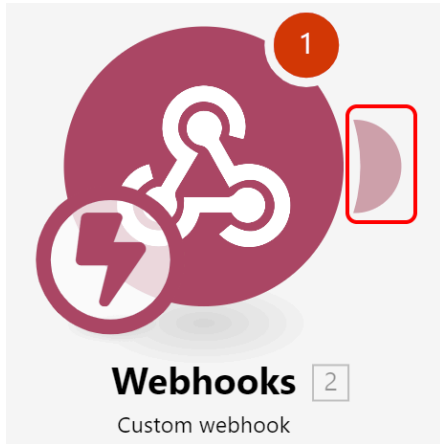


- P. Go back to **GW Apps**, and open a record from the appropriate form that is at the workflow stage where you added the 'Test Webhook' workflow action button. Click on 'Test Webhook'. The screen will refresh but no message or other indication will display to confirm it was successfully triggered.
- Q. Go back to **Make**, and you should see a messenger that the webhook successfully acquired the test data. Depending on how quickly you switch between the browser tabs and on system load, it might take a few seconds to display after you are back at Make.

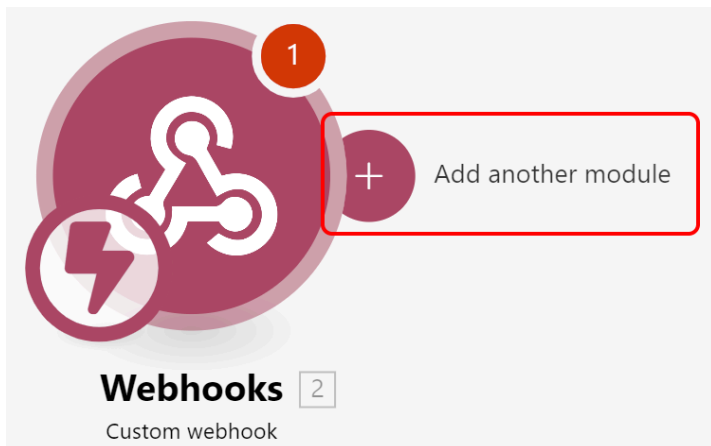


We now have a working webhook. Next, we need to create an HTTP Request to parse the data received by the webhook, authenticate with GW Apps and receive an authentication token. After that, we will create a second one to post data back to GW Apps via the GW Apps API.

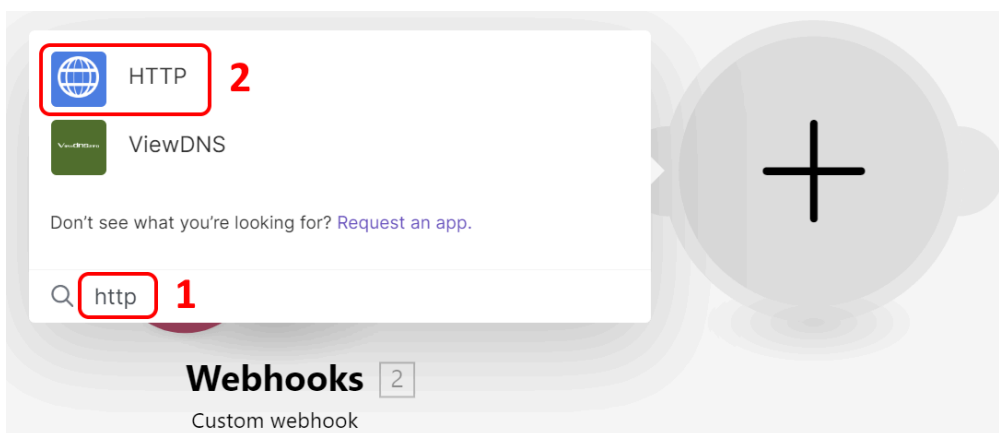
R. Click on the crescent shape to the right of the 'Webhooks' module circle:



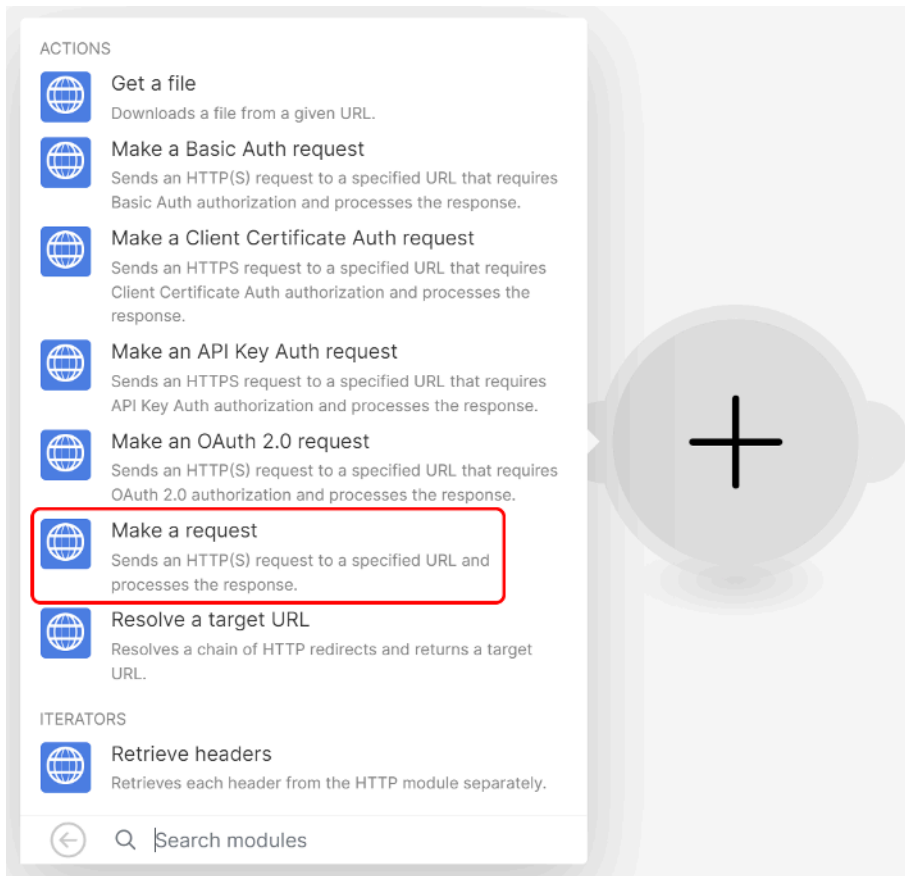
The crescent will change into an '+ Add another module' option. Click on '+ Add another module'.



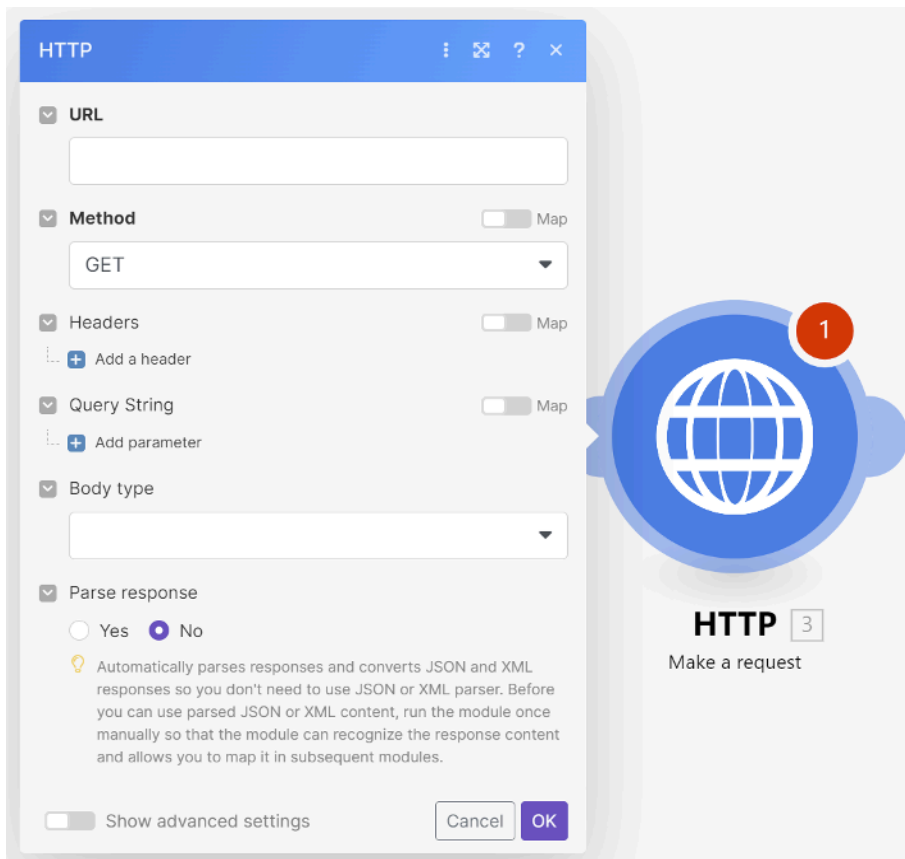
S. Type "HTTP" in to the search field at the bottom of the module selection dialog (1), then click on HTTP at the top of the list (2):



The dialog will change to display the HTTP request options. Select 'Make a request':



T. You will see the HTTP Request configuration dialog:



Parameter	Value
URL	https://api.gwapps.com/v1/token
Method	POST
Body type	Raw
Content type	JSON (application/json)
Request Content (Type as a single line of text)	{"key": "- API Key of the app -", "email": "- email address -", "customerId": "- Your Customer ID -"}
Parse Response	Yes

As shown above, you will need three parameters that are specific to your app, you as a GW Apps client and to the user whose ID will be making the request. Below we explain how to acquire these values:

- API Key of the app -

You need to create an API key for use with this Make exercise. It is possible to create an API Key with full access to everything in an app, and then use that in multiple scenarios. However, that API key would have significant access to your app and data. Instead, it is recommended to make specific, tightly controlled API keys for each scenario so that if a key is compromised it will limit the security exposure. Detailed help on creating API Keys is covered in the [Security - API Key Configuration](#) support site article. Here we will only cover the details required for this exercise.

Go back to **GW Apps**, and go to the Test App. Open the API Keys section (Edit App > Security > API Keys) and create a new API Key (+ Create API Key at the top-left of the main area, give it a suitable name and click on 'Create':

New Api Key
×

Api Key Name *

8 / 128

Description

Cancel Create

Generated API Key	Copy this key and save it in a secure place. You will not be able to see this key again once this form is saved.
Security Details	<p>Choose 'All Domain Users' or 'Specific Users'. (For security reasons, it is strongly recommended to use specific users.) For 'Specific Users' select the desired users in the field to the right.</p> <p>For this exercise, use 'Specific Users' and add the username for the user you are planning to use in the 'email' parameter in the API call.</p>
Access Details	<p>Each of the forms in the application will be listed, and they will all have all access scopes selected. For security reasons, it is strongly recommended to remove any forms not related to this specific scenario, and to also deselect any scopes not required by this scenario.</p> <p>For this exercise, make sure the form with the text/text area field is listed and leave all the scopes selected.</p>

Api Key Configurations: Make.com API Call

Generated API Key: 🔗 MAKES-SECRET-KEY-XXXXXXXXXXXX 📄

ⓘ Store the generated api key in a safe place, once this form is closed you will not be able to see it again.

General Information

Name *
Make.com API Call

Description

Security Details ?

Type *
Specific users

richard.knight@gwapps.com ✕ Type user's name or email 🔍

Access Details ?

Form
Text Area

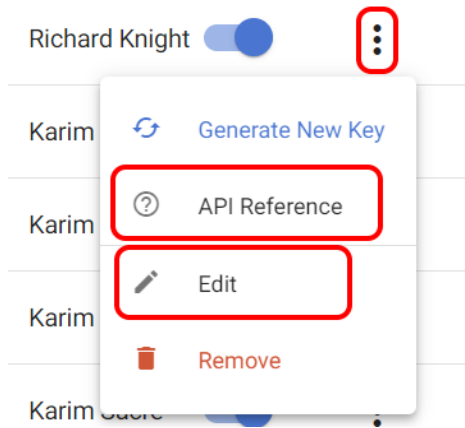
+ Add Access

- Create Records
- Update Records
- Delete Records
- View and Search Records
- Export Records to PDF
- View Form

Save

Double check you have the API key copied to the clipboard or saved somewhere, then click 'Save'.

Now that you have an API Key, you can edit it or regenerate the key value by clicking on the three-dot menu at the end of the row for the API key, Also here is the API Reference. This provides detailed examples of API calls and also provides access to your Customer ID and the form's ID (assuming the key is only scoped to one form).



- email address -

Simply add the email address of whichever user you want to access GW Apps in the API call. Make sure the user has sufficient permissions on the required records at the workflow stage they might be at, and if required, are named in the API key.

- Your Customer ID -

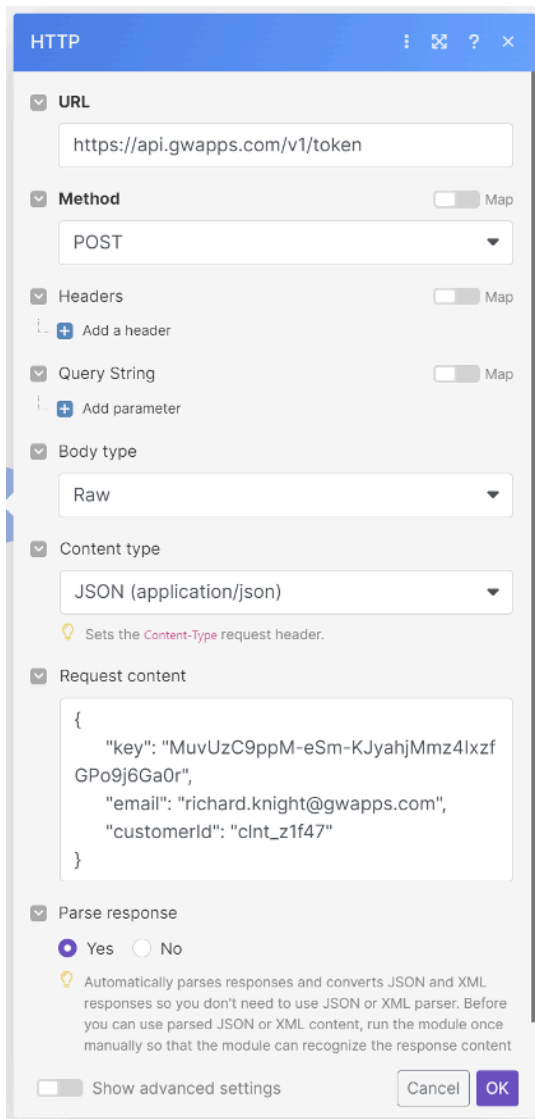
Select the API Reference from the three-dot menu, and then click on 'Authentication' in the left navigation. Your specific Customer ID will be displayed in the two locations circled in red below.

Authentication
Your application must use this API "/>

U. Go back to **Make**, and complete the Request Content, using the API Key and Customer ID values you just created/looked up. Do not use the values shown below, these are just examples. Use the values for your API Key, customer id and user email.

```
{
  "key": "MuvUzC9ppM-eSm-KJyahjMmz4lxzfGPo9j6Ga0r",
  "email": "richard.knight@gwapps.com",
  "customerId": "clnt_z1f47"
}
```

The completed HTTP Request should look similar to the following (with different key, email and customerid values):



Click on 'OK' to save the configuration.

- V. We need to create another HTTP Request to use the API to update the record in GW Apps with the new text value. Repeat step S to create a second HTTP Request. The configuration settings for this second HTTP Request are shown below:

Parameter	Value
URL	https://api.gwapps.com/v1/forms/<form_id>/records/<record_id>
Method	PUT
Headers	> Name: Authorization > Value: <token_type> <access_token> Note: Make sure you have the single space character between the two values.
Body type	Raw
Content type	JSON (application/json)
Request Content	{"<field_shortcode>": "<text value>", "stage": "<webhook_stage>"}
Parse Response	Yes

URL

To get the <form_id>, edit the desired form. The <form_id> is part of the current page URL in the browser. The URL will have the following format:

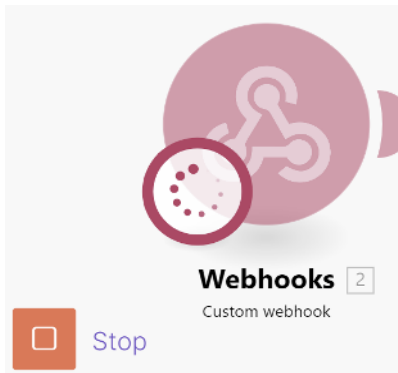
https://app.gwapps.com/<app_id>/designer/forms/<form_id>/design

A specific example:

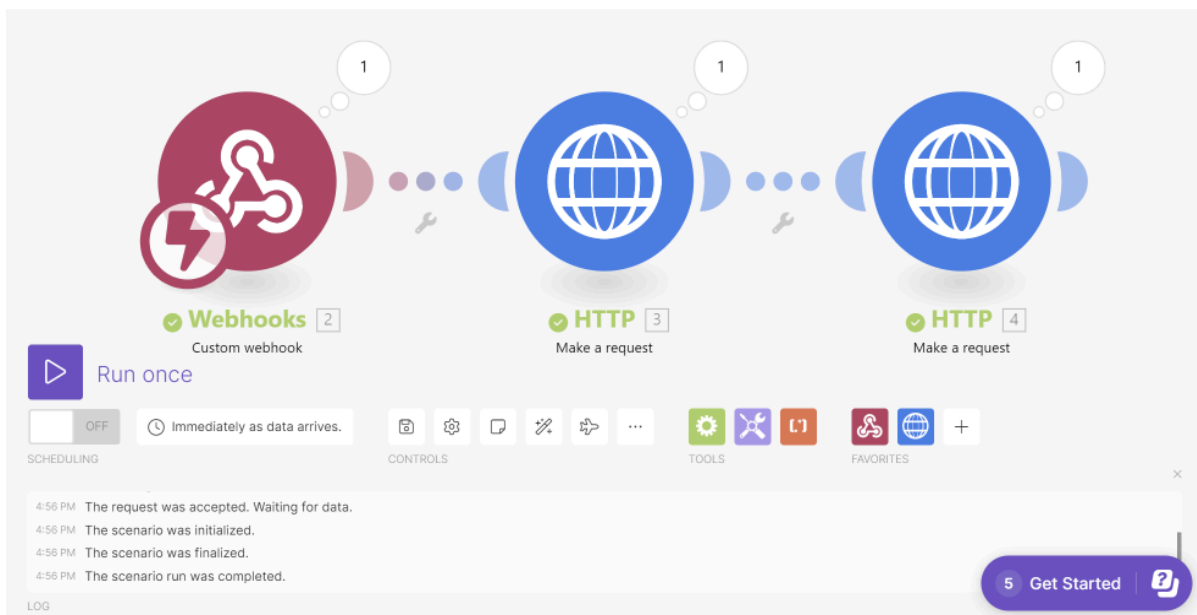
https://app.gwapps.com/5d72ee95fabe1f0371f19455/designer/forms/649a8d9557d9e7fed520f82d/design

Copy the <form_id> and add it to the URL string. For the moment you will not know the <record_id> or the <token-type>, <access_token> or some of the values for the Request Content field. That is OK for now: We need to do this in steps not all at once. We will need to run the webhook one time so that the first HTTP call can parse those values out for us, so that we can then use them in the second HTTP request. So for now, literally leave them as "<record_id>", "<access_token>" and so on. This does mean the update request will fail, but that is OK as we don't even expect it to work yet.

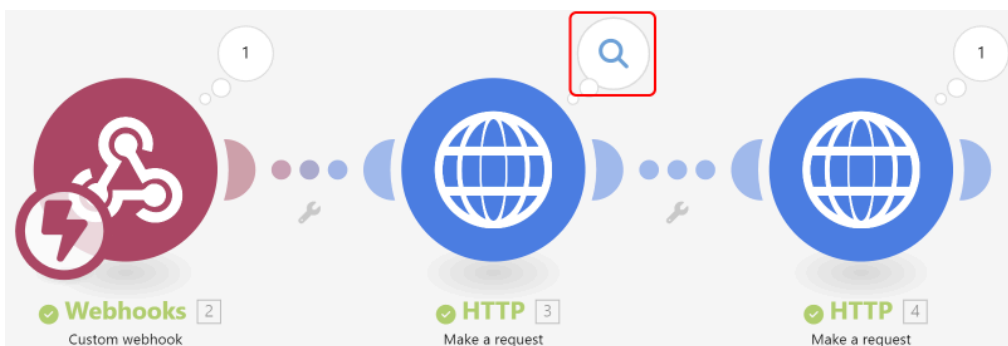
W. Click on the 'Run Once' button: You will see the webhook module show it is waiting and an orange 'Stop' button at the bottom:



Now go back to **GW Apps**, open the Test App, then open a suitable test record and click on the workflow button that triggers the webhook. Then go back to **Make** and you should see the screen update to show that it has successfully run the scenario and parsed the data returned by GW Apps. Note green labels for the three modules and the short list of log entries showing it ran.



X. Now we can check that the parts that could run properly did run, and update the final HTTP request to use the data values coming from the webhook and the first HTTP request.



If you hover over any of the bubbles above and right of the modules, the bubble will change to showing a spyglass. Clicking on it will return the log details for that module in the last run of the scenario.

The screenshot displays an HTTP client interface. On the left, a large blue globe icon is visible, with a smaller bubble containing a magnifying glass icon above it. Below the globe, a green checkmark and the text 'HTTP 3' are shown, along with the instruction 'Make a request'. The main panel on the right is titled 'HTTP' and shows the details of a request and response. The request section is expanded, showing the following details:

- Initialization
- Operation 1 ▲ (Data size: 2.1 KB)
- INPUT
 - Bundle 1: (Collection)
 - Self-signed certificate: *empty*
 - Query String: (Array)
 - URL: `https://api.gwapps.com/v1/token`
 - Request content: `{ "key": "MuvUzC9ppM-eSm-KJyahjMmz4lxzfGPo9j6Ga0r", "email": "richard.knight@gwapps.com", "customerId": "citen_e0b70" }`
 - Request compressed content: *true*
 - Method: *post*
 - Headers: (Array)
 - Timeout: *empty*
 - Use Mutual TLS: *false*
 - Password: *empty*
 - User name: *empty*
 - Body type: *raw*
 - Content type: *application/json*
 - Serialize URL: *false*
 - Share cookies with other HTTP mod...: *false*
 - Parse response: *true*
 - Follow redirect: *true*
 - Disable serialization of multiple sam...: *false*
 - Follow all redirect: *false*
 - Reject connections that are using u...: *true*
- OUTPUT
 - Bundle 1: (Collection)
 - Status code: **201**
 - Headers: (Array)
 - Cookie headers: (Array)
 - Data: (Collection)
 - access_token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyYjVvKThjZmFqRW9kGyRxcASWUQ`
 - expires_in: *3600*
 - token_type: *Bearer*
 - fileSize: *482*

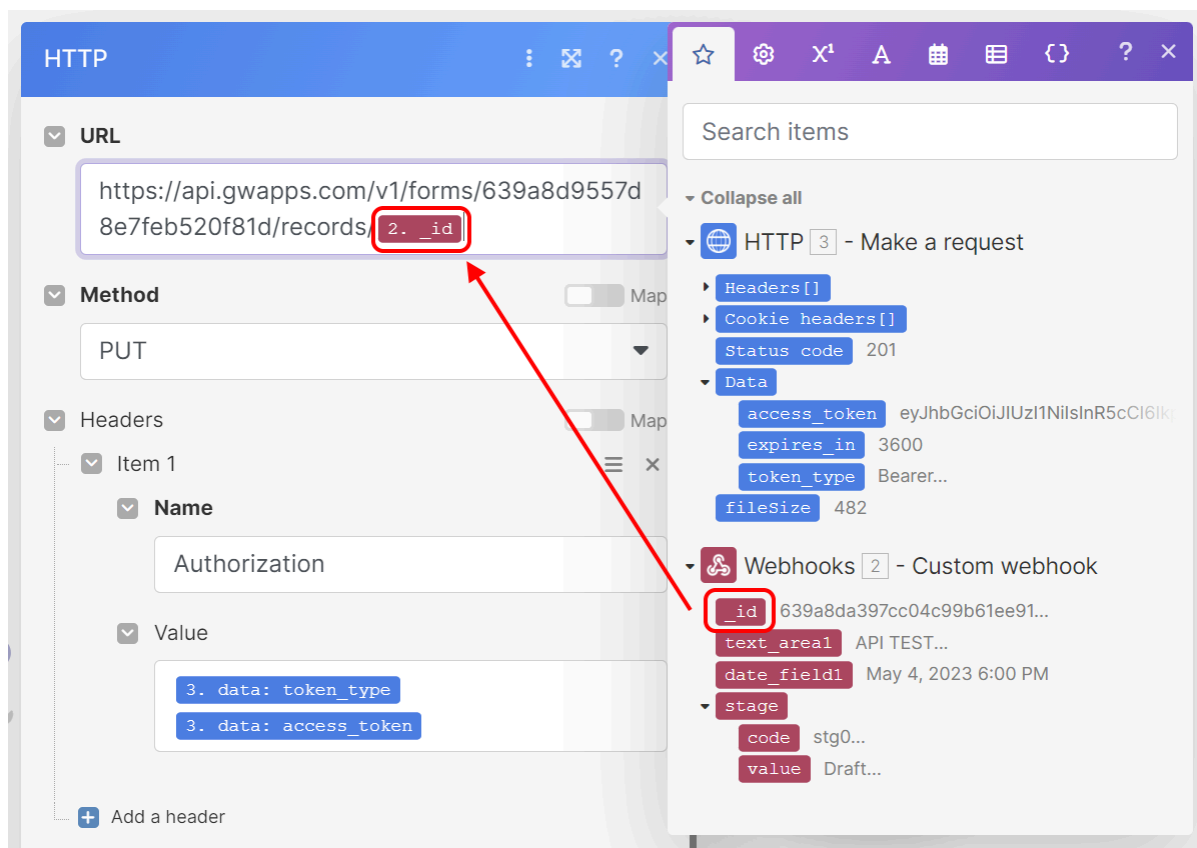
At the bottom of the interface, there are icons for 'TOOLS' (gear, wrench, terminal) and 'FAVORITE' (refresh, share). The status bar at the bottom shows 'Commit' and 'Finalization' with green checkmarks.

As you can see, the output includes a Status code of '201', which means it ran OK, and we have the expected access_token being returned. If you have a Status code of anything but '201', please carefully check the configuration of the first HTTP request, fix any mistakes, and run it again.

Now, click on the module for the second HTTP request. We can now finish it's configuration.

<record_id>

When you click in any of the configuration fields, such as URL or Request content, you get the same helper dialog box appear, but now it has useful values in it. In the example below you can see that the webhooks section (2 - this number is simply the numeric ID of the module, it is the same number that displays below the modules when you are looking at the whole scenario) has an '_id' value available. This is the value for the <record_id> that we need. So, click into the URL field, delete the '<record_id>' dummy text that you had, and click on the '_id' in the dialog. You will now see a red rectangle with '2._id' in it at the end of the URL string. (the "2" is just the ID for the webhook module, as mentioned above.) Now when the second HTTP request is run, the value of the <record_id> will be spliced into the URL before it is actually called.



<token_type> <access_token>

We need to do the same to the value of the Authorization header. Clear the text in the value field, and select 'token_type' from the blue HTTP request section of the dialog, add a single space and then select 'access_token' from the blue HTTP request section of the dialog.

Note: Make sure you have the single space character between the two values.

The screenshot displays an HTTP client interface with the following configuration:

- URL:** `https://api.gwapps.com/v1/forms/639a8d9557d8e7feb520f81d/records/2._id`
- Method:** PUT
- Headers:**
 - Name:** Authorization
 - Value:** 3. data: token_type 3. data: access_token
- Body type:** Raw

The response data on the right shows the following structure:

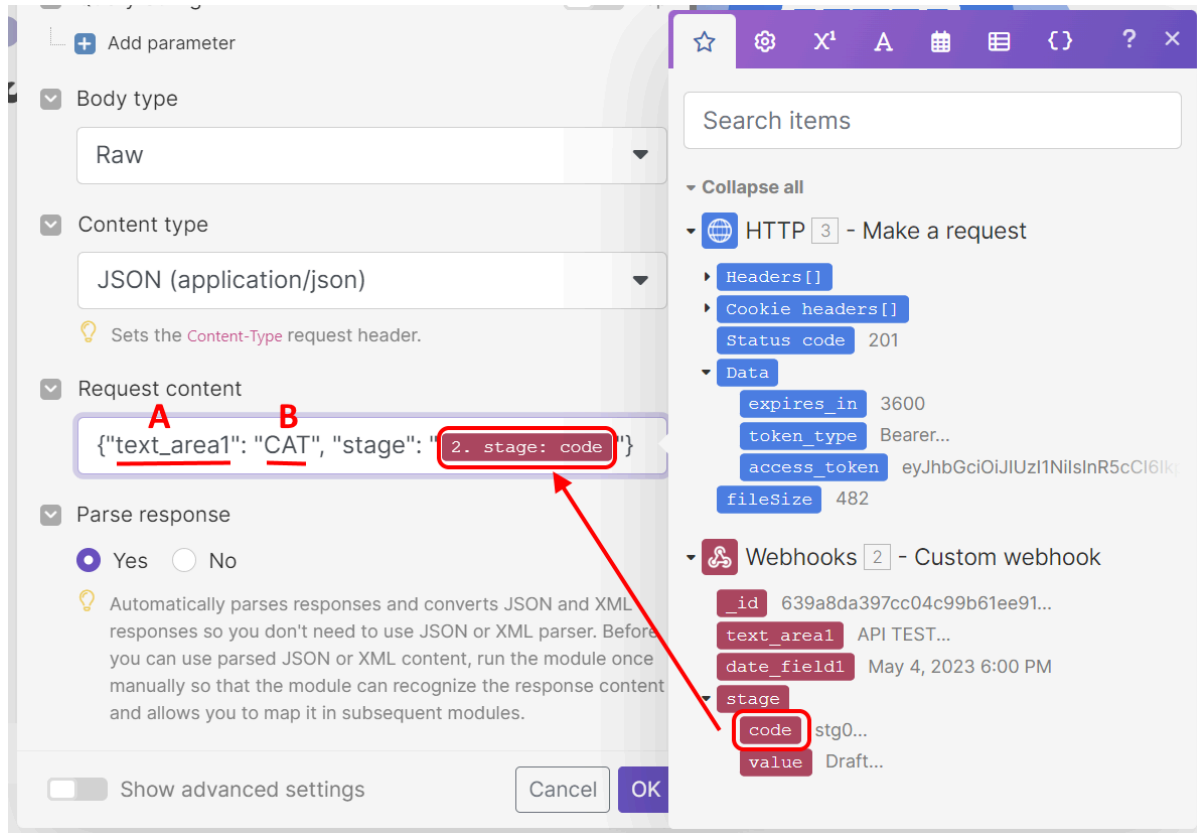
```
{
  "expires_in": 3600,
  "token_type": "Bearer...",
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6Ikpz...",
  "fileSize": 482
}
```

Red boxes highlight the 'token_type' and 'access_token' fields in the response data, with red arrows pointing to the corresponding values in the Authorization header value field.

Request Content

Finally, we need to complete the Request Content. At the moment yours should have the following value:

```
{"<field_shortcode>": "<text value>", "stage": "<webhook_stage>"}
```

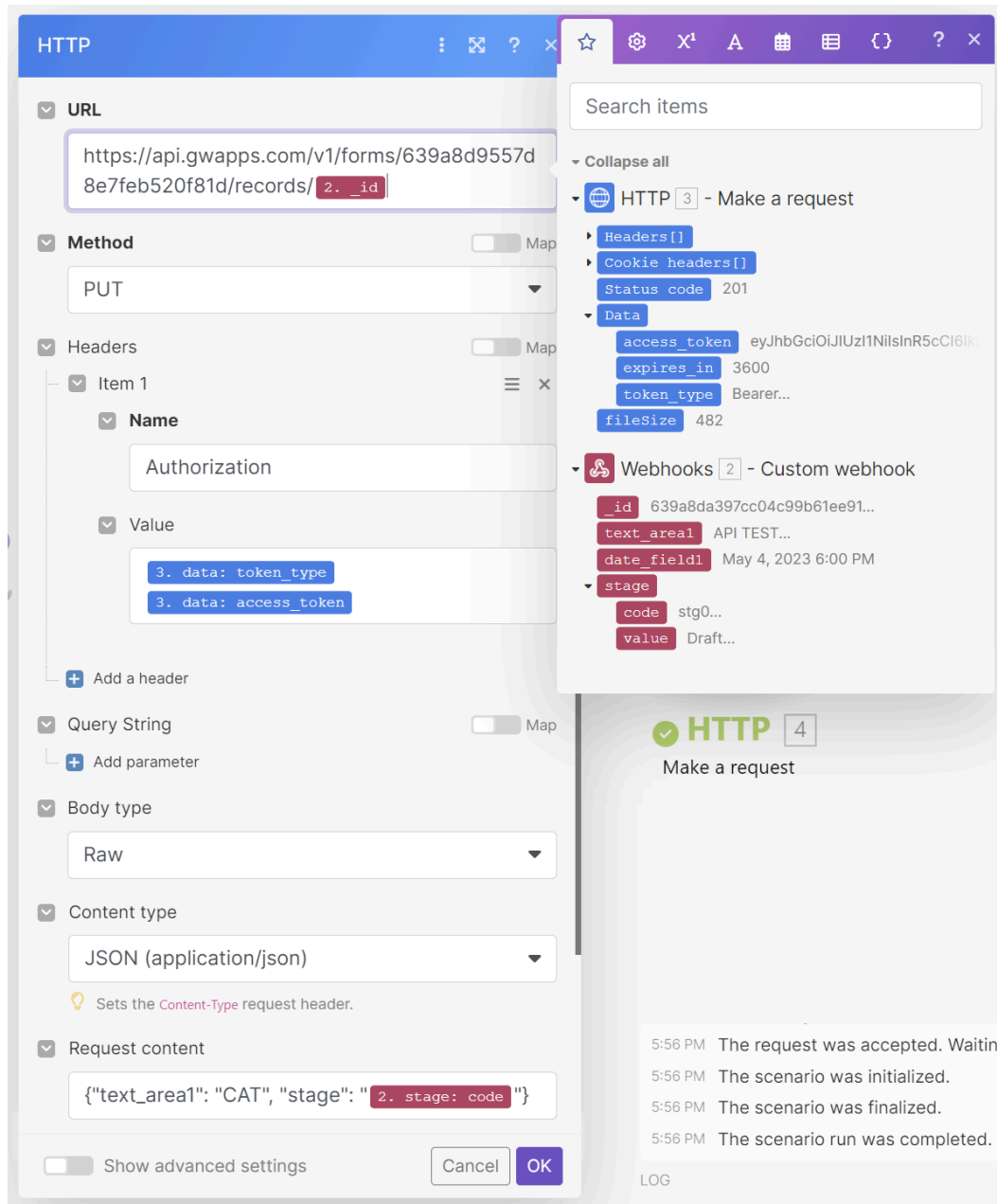


Delete '`<field_shortcode>`' from the Request content field (A), making sure to keep the double quotes that were around it, and then with the cursor between the double quotes type in the shortcode of the text field on your form.

Delete '`<text_value>`' from the Request content field (B), making sure to keep the double quotes that were around it, and then with the cursor between the double quotes type in the text you would like the API to put in the field: In this example "CAT".

Delete '`<webhook_stage>`' from the Request content field, making sure to keep the double quotes that were around it, and then with the cursor between the double quotes, select 'code' from under 'stage' in the red Webhooks section of the dialog.

You should now see something like the following:



Y. You should now be able to successfully run the webhook and have the value updated in the record when it is complete. Click on 'Run Once' and then go back to **GW Apps**, open the Test App and a suitable test record, and click on the workflow button that triggers the webhook. Then go back to **Make**, where you should see the screen update to show that it has successfully run the scenario. Check the logs for both of the HTTP requests, by clicking on the bubble above and right of the module, and check both of them have a '201' status code. If they did, then go back to your test record in GW Apps and see if the field has been updated to the value you specified in the second HTTP request.

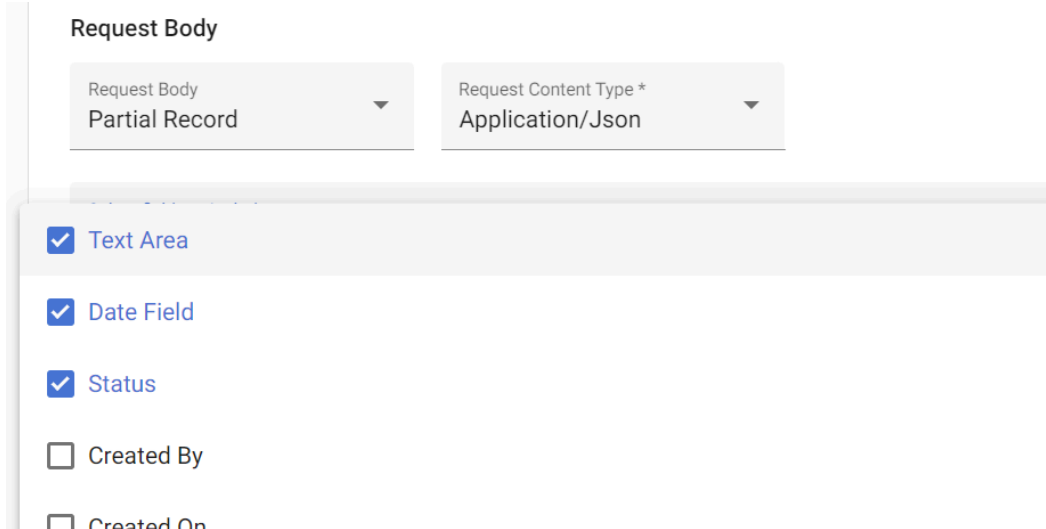
If not, carefully check the configuration of the failing module and try again.

Part 2: Make Data Manipulation Functions

The dialog box that shows the workflow stage code (2. stage: code) and access token (2._id), has several tabs at the top. These give you access to functions for handling generic actions, math, text, date/times and arrays. These can allow you to do things like add a piece of text before or after the current text in a field, or add three days to the date stored in a field. The documentation in Make can help with using these more advanced features, and many others we have not covered in this document.

As an example, the steps below show how you can use the date functions in Make to read a date field, add 5 days to the date value, and add that into the text in the Text or Text Area field. You will need to add a Date field to the form you are using for this exercise. In our form, the field shortcode for the date field was "date_field1".

You will need to go back and edit your webhook Action in GW Apps. You will need to update the 'Select fields to include' field from the Request Body section to include your new date field.



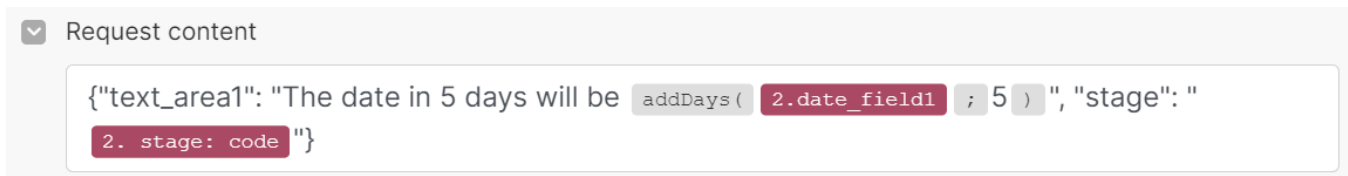
Request Body

Request Body: Partial Record

Request Content Type *: Application/Json

- Text Area
- Date Field
- Status
- Created By
- Created On

You will then need to repeat steps O, P and Q to allow the Webhook module in Make to parse the new date field from the incoming request. You will now be able to access that date field in the second HTTP Request. We will update the Request content to look like this:



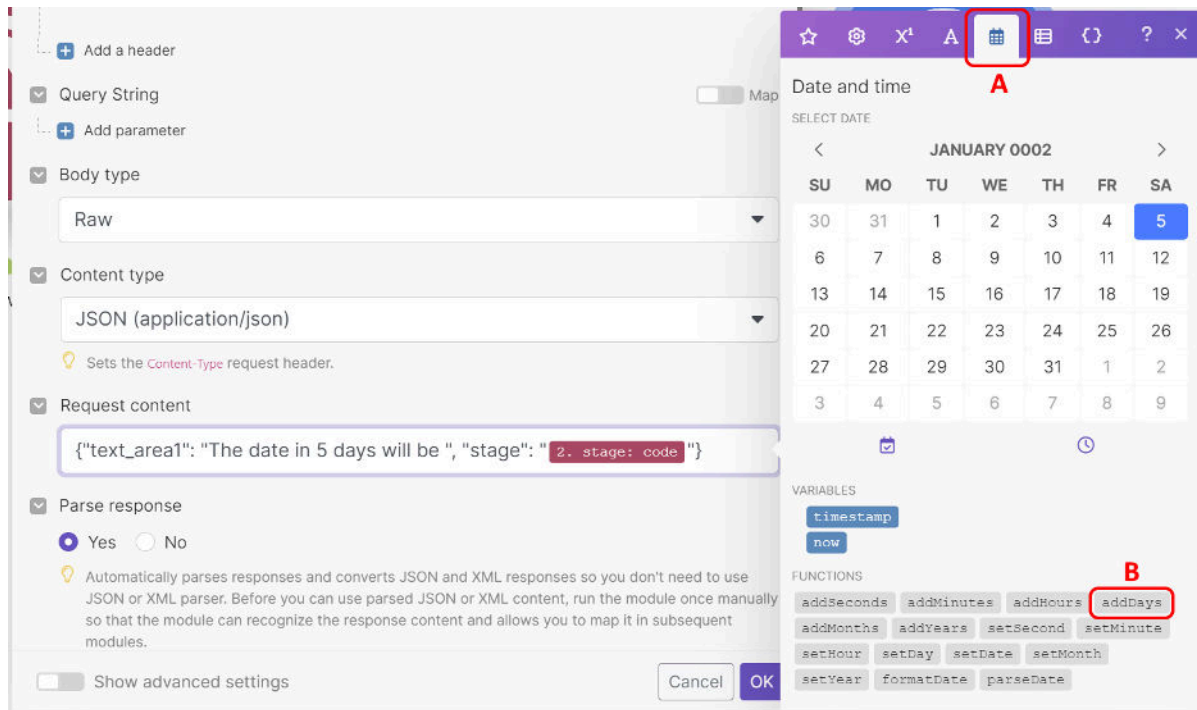
Request content

```
{ "text_area1": "The date in 5 days will be addDays ( 2.date_field1 ; 5 ) ", "stage": " 2. stage: code " }
```

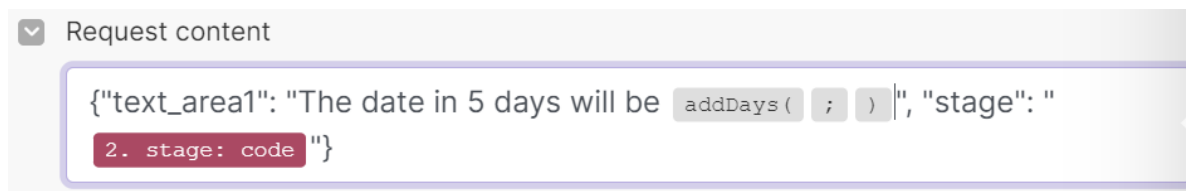
Entering the new date formula is a little fiddly the first time, but it is easy to get the hang of how to build them. I will step through building the formula in detail:

- Delete the current text value within the double quotes: 'CAT', unless you chose to use different text while doing the main exercise.

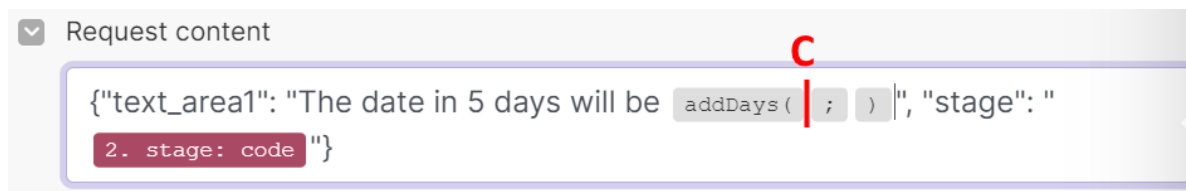
- Within the double quotes add 'The date in 5 days will be '. (Do include the space after 'be')
- Make sure your cursor is still after the 'be ' and just before the closing double quote. In the dialog box, select the 5th tab at the top with the calendar icon (A). Then click on the 'addDate' function (B).



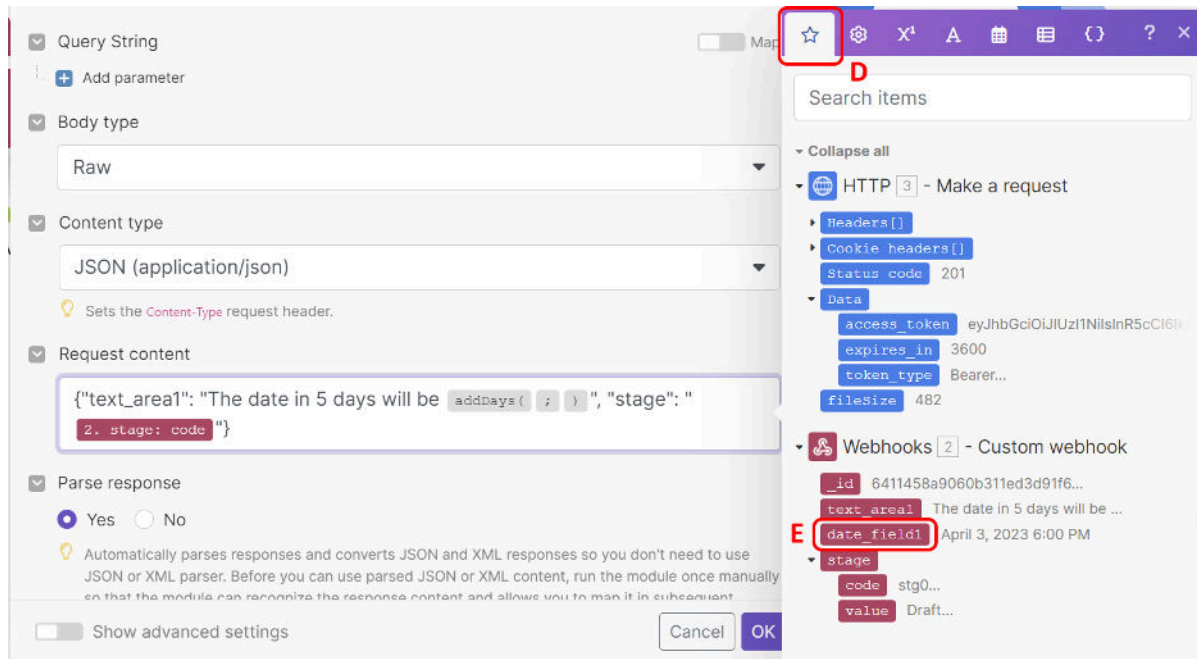
The Request content should now look like this:



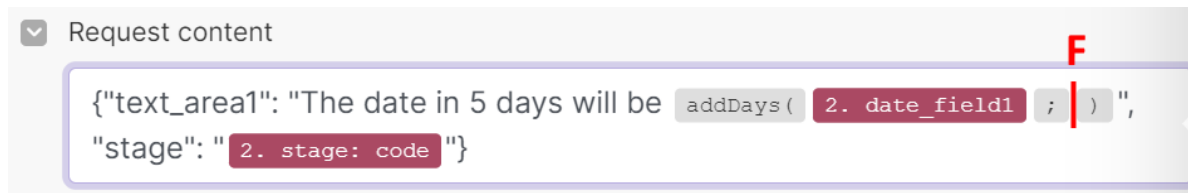
- Place your cursor in the white gap between the gray 'addDays(' and the gray ';' (C). This is the space for the first parameter of the addDays function.



- In the dialog box, select the 1st tab at the top with the star icon (D). Then click on the 'date_field1' field from the Webhooks section (E).



- Place your cursor in the white gap between the gray ';' and the gray ')' (F). This is the space for the second parameter of the addDays function.



- Type in '5'.
- Now save and run the webhook again. (Save the scenario, click on 'Run once', then go back to **GW Apps** and open your test record, make sure a date value is present in the date field and click on the webhook action button. Go back to **Make** and check that the scenario ran successfully. If it did go back to **GW Apps** and refresh your browser so that the screen is updated to show the change in value. You should see something like the following:

Text Area

The date in 5 days will be 2023-04-09T00:00:00.000Z

Date Field

Apr 4, 2023

- The date is being displayed with all date-time data elements and in the UTC+0/GMT time zone. If you would like to have the date formatted more neatly, and displayed in a specific time zone, you can add the formatDate function to achieve that. Place your cursor immediately before the 'addDays(' function opening, and select 'formatDate' from the 5th tab of the dialog. Now copy the whole of the current addDays function 'addDays(2.date_field1; 5)' and paste it

into the space for the first parameter of the new `formatDate` function. Type 'MM/DD/YY' into the space for the second parameter. Make only shows the first two parameters initially, but we need to use the optional third parameter to set our desired time zone. After the '/YY' type a semicolon ';' which should turn gray once you have typed it. Now add 'America/Los_Angeles', or another valid time zone value, into the newly created space for the third parameter. The final Request content should look like this:

```
Request content
{"text_area1": "The date in 5 days will be ", "stage": "2. stage: code"}
formatDate( addDays( 2.date_field1 ;
5 ) ; MM/DD/YY ; America/Los_Angeles )
```

- Now save and run the webhook again. (Save the scenario, click on 'Run once', then go back to **GW Apps** and open your test record, make sure a date value is present in the date field and click on the webhook action button. Go back to **Make** and check that the scenario ran successfully. If it did go back to **GW Apps** and refresh your browser so that the screen is updated to show the change in value. You should see something like the following:

Text Area

The date in 5 days will be 04/08/23

Date Field

Apr 4, 2023

Webhook / API Use Case Ideas

This simple example was just to get you used to the process of using Make to interact with GW Apps via webhooks and API calls. Now that you have an understanding of the process, you can easily extend this base to do more complex things with GW Apps and Make. Here are a few examples our customers have implemented or discussed:

- Integrate GW Apps and Google Calendar so that events created in a custom GW Apps event management app can be automatically created in Google Calendar. Updates to the event record can also update the Google Calendar entry automatically.
- Implement complex mathematical calculations not currently supported in GW Apps, both for numbers and date/time values. For example you could do complex financial or technical calculations.
- Create Slack posts automatically when a record gets to a specific stage in a workflow. For example, in a Product Feature Management app you could post to the slack marketing channel that the new feature was approved and allow them to start thinking of sales and marketing avenues related to this new feature. (You could also post messages to Facebook, Google Chat and many other platforms.)