

Postman

API Development Collaboration

- Agenda

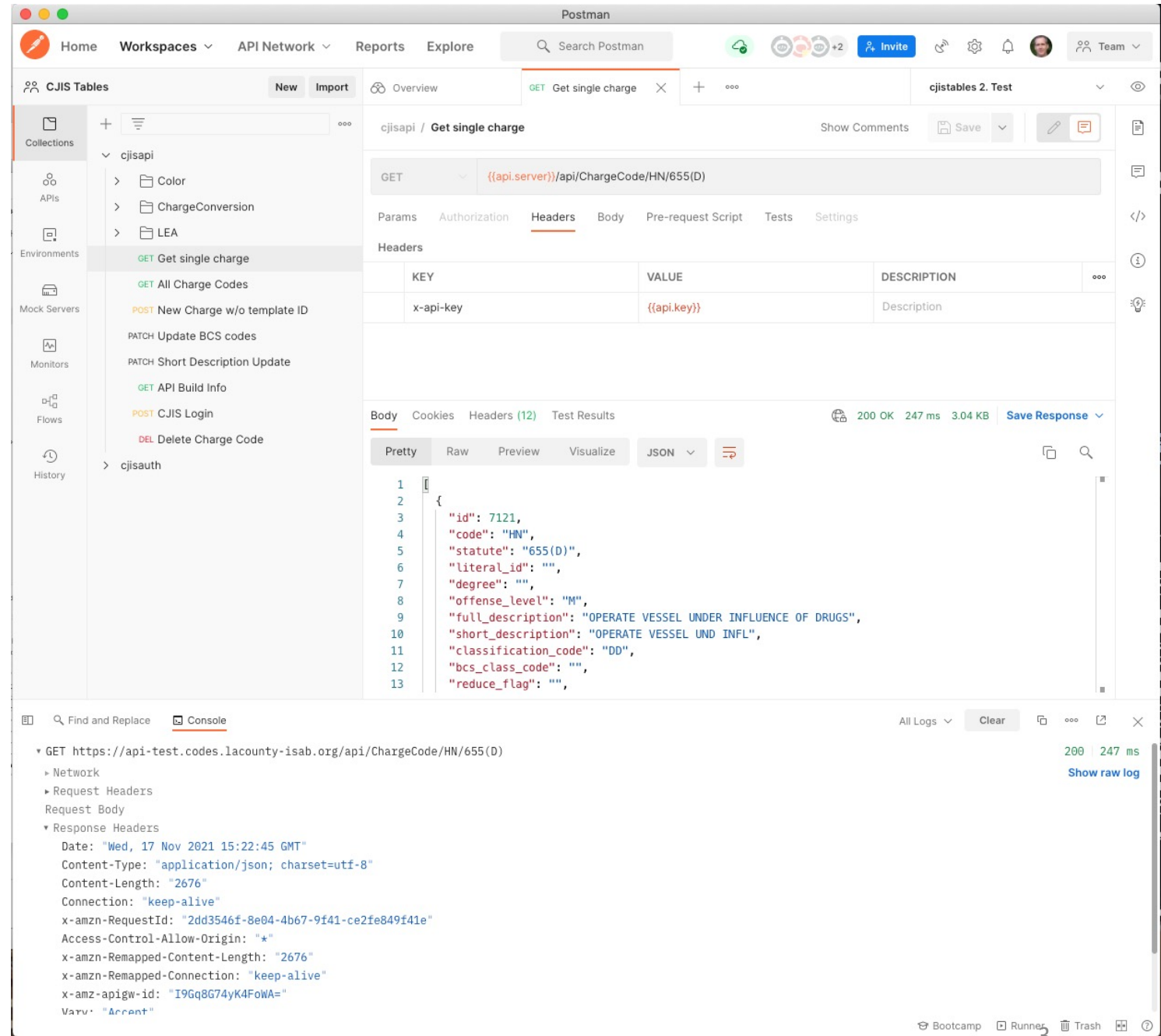
- Postman Overview
- Native App vs Web
- Workspaces
- Collections
- Environments
- Variables
- Scripts

- Assumptions – you

- are involved with API development at some level
 - provider
 - consumer
 - analyst
- understand the role that Postman plays in API testing
- understand the difference between a web application and a workstation application
- have a Postman account

What is Postman

- A Web API testing tool
 - Nice UI
 - Configure all details of a request
 - Inspect all details of a response
 - Organize requests
 - Multi-environment support
- Other features
 - Mock Servers
 - OpenAPI interface specifications
 - Team Support



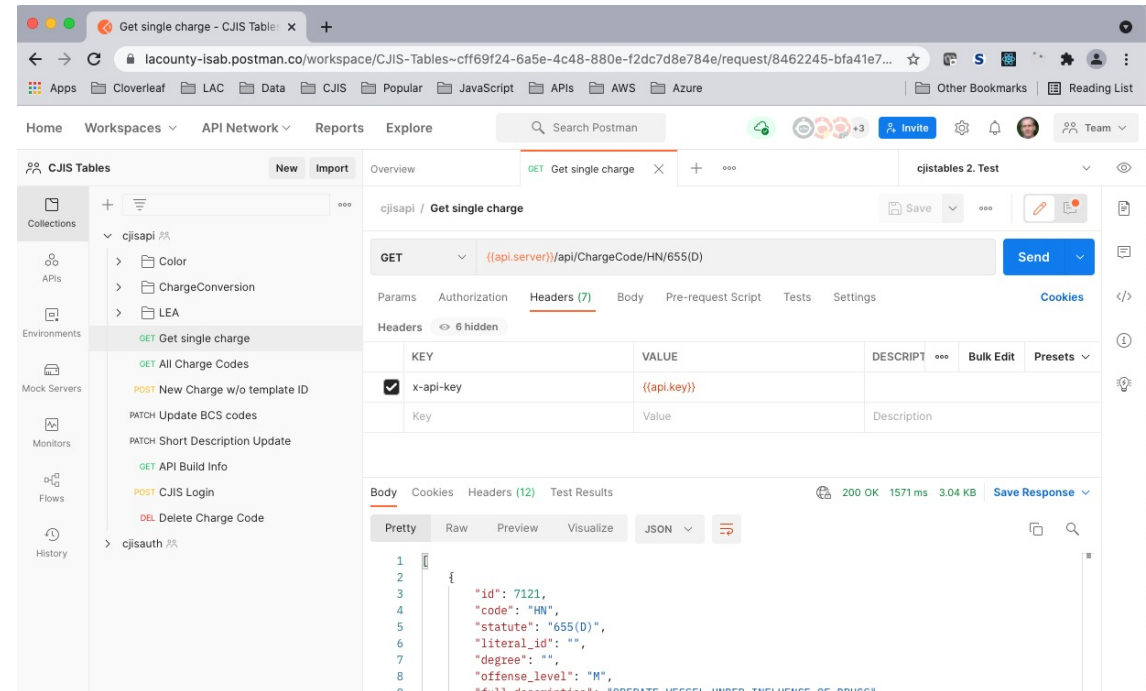
Native vs Web Application

The native and web Postman applications are almost identical. The primary difference is that the web application runs in a browser tab. As such, the web application is prone to

- getting lost among other browser tabs
- accidentally navigate away from the app
- accidentally close the tab or even the browser

The native application has direct access to your machine's file system and network sockets.

But the browser has the advantage of being better integrated with network perimeter security.



Using Postman as a web application in a browser tab

Quick Dive

The quickest way to get a feel for the Postman UI is just to use it for some simple scenarios.

Exercise 1 – Workspaces and collections

Exercise 2 – Making changes

Exercise 3 – Managing environments

Exercise 4 – Managing Headers

Exercise 5 – Documentation and Comments

Exercises 6 – 10 – Full Scenario

The exercises use a pair of API keys and a sample account. These values are located in the MS Teams **Postman** channel. Select the **Info** wiki and then the **Credentials** page. It might help if you open that right now for reference.



Photo by [Ryoji Hayasaka](#) on [Unsplash](#)

Exercise 1 – Workspaces and Collections

The goal of this exercise is to quickly obtain hands-on experience with the fundamental Postman concepts of a *workspace* and a *collection*.

1. Open Postman, either the native application or the web application.
2. Create a new workspace by selecting the **Workspaces** dropdown and selecting **Create Workspace**.
 - a) For **Name** enter ISAB Workshop 1
 - b) For **Summary** enter: First workspace for Postman workshop
 - c) For **Visibility** select **Personal**.
 - d) Click **Create Workspace** button. This should create a new empty workspace.
3. Underneath the summary, click your mouse in the **Description** area. When empty, it contains placeholder text *Add a description ...*. Add the following description and click **Save**.

Used for **private** workshop experimentation. Will likely be ****deleted**** after workshop. Markdown descriptions support

- * bullet lists
- * ``monospace font`` entries
- * [links to other pages] (<https://workshops.lacounty-isab.org>)

and other basic formatting.



Reality Check – Markdown

Markdown has become the de facto formatting syntax for web documentation. For more information on markdown and its syntax, see the following links.

- <https://daringfireball.net/projects/markdown/basics>
- <https://guides.github.com/features/mastering-markdown/>

Exercise 1 – Continued


4. Select **Collections** from the left navbar and click **Create Collection**. Name the new collection **Samples**.
5. Click the "..." next to the collection name in the collection hierarchy and select **Add folder**. Name the new folder **Postman** and add, as documentation, "Sample APIs hosted by Postman".
6. Click the "..." next to the new folder and select **Add Request**. Name the request **Echo** and leave the method as **GET**. enter a request URL of `https://postman-echo.com/get`.
7. Click the big blue **Send** button to invoke the sample API. It should return the request header values as a JSON object. Save the request.
8. Using the same process, create another top-level folder called **CJIS Tables**.
9. Add a new request named *Get charge 826*.
10. Leave the method as **GET** and add a URL of `https://api.codes.lacounty-isab.org/api/ChargeCode/826`.
11. Click **Send**. The HTTP response code should be *403 Forbidden*. We'll fix this later. Save the request.

In this exercise you

- created a new personal workspace
- created a new collection
- created multiple folders
- created multiple GET requests
- observed different HTTP status codes
 - 200 – "success"
 - 403 – "forbidden"

Exercise 2 – Changing Environments

This exercise continues where the previous exercise ended – with the *Get charge 826* request.

1. Change the URL request field of the request to invoke the TEST API instead of PROD by changing `api.codes` to `api-test.codes` in the URL field.
2. Send the request with the **Send** button. You should see the same 403 response as you did for PROD.
3. Select the **Auth** tab in the request section for adding an API key. The type defaults to *Inherit auth from parent*. In this case, the parent is the **CJIS Tables** folder. Since the same API key will work across all requests, it makes sense to configure authorization at the folder level to avoid duplicating it across several requests.
4. Select the parent folder, **CJIS Tables**, and the **Authorization** tab within it. Notice that it, too, is configured to *Inherit auth from parent*. But since this authorization doesn't apply to other siblings, we'll configure the authorization here.
5. From the dropdown box, select **API Key**.
6. For the **Key** field, enter `x-api-key`
7. For the **Value** field, enter the TEST API key. This will add the API key to the header.
8. Save the folder settings.
9. Go back to the *Get charge 826* request tab and send the request. It should be successful.
10. Now change the URL back to the PROD URL, i.e. change `api-test` back to `api` and try again. 

The request should fail. That's because we still have the TEST API key configured in the folder authorization. Changing environments is no longer as simple as changing the URL. Now we have to change the URL **and** the authorization. As our API becomes more realistic, these environment differences grow cumbersome to swap. **Environment variables** address this.

Exercise 3 – Managing Environments

In the previous exercise, we saw that switching between environments can become cumbersome as the number of environmental dependencies grows. This exercise will demonstrate how **Postman environments** address this.

1. Select **Environments** from the left navbar. So far, we should have no environments defined.
2. Click the plus symbol, **+**, at the top-left of the (empty) environment list to create a new environment.
3. Name the new environment **Test**. It is displayed as a table of values.
4. For the first row of the table, make the following entries:
 - a. **Variable:** `baseUrl`
 - b. **Initial Value:** `https://api-test.codes.lacounty-isab.org`

The Current Value should be filled automatically with the Initial Value. If not, make sure it is.

5. For the second row, add the following entries:
 - a. **Variable:** `apikey`
 - b. **Initial Value:** `Your API Key`
 - c. **Current Value:** `<Test API key>`
6. **Save** the environment.

You can see that each Postman environment variable contains two values:

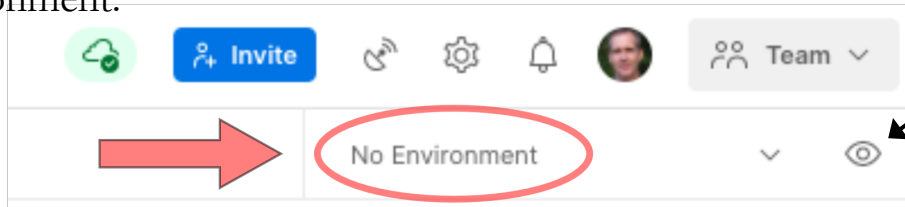
- **Initial Value** – visible to all members of the workspace
- **Current Value** – used by **your** workspace

Test		Your team sees this	Only you can see this
	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	baseUrl	https://api-test.codes.lacounty-isab.org	https://api-test.codes.lacounty-isab.org
<input checked="" type="checkbox"/>	apikey	Your API key	

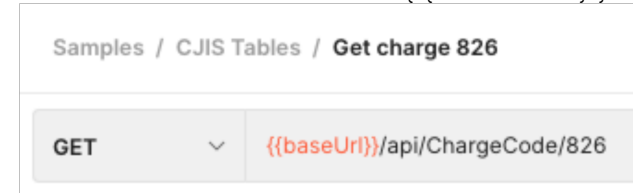
Exercise 3 (continued)

7. Create a new environment with the **+** button again.
8. Name the new environment **Production**.
9. Add the **baseUrl** variable with an initial value of `https://api.codes.lacounty-isab.org`.
10. Save the environment (we're not adding an API key yet).
11. Close the **Test** and **Production** environment tabs from your workspace and select the **CJIS Tables** folder. It should still have the **Authorization** section selected.
12. Change the **Value** field to `{{apikey}}`.

At this point, the value should appear in red. This is because it's not defined for the current environment. In fact, there is no current environment.



13. From the environment list in the upper-right corner, select **Test**. At this point, the `{{apikey}}` value you provided in Step 12 should turn from red to orange. This indicates this value can be read from the current environment.
14. Save the folder tab and select the *Get charge 826* tab.
15. Change the URL field to contain the `{{baseUrl}}` variable.



16. Now send the request. You should see the result from the CJIS Tables TEST environment.
17. Select **Production** from the environment list.
18. Send another request to verify the request fails due to the fact we didn't set the API key for this environment.
19. Verify the environment values by clicking the "eye" icon.

Production			Edit
VARIABLE	INITIAL VALUE	CURRENT VALUE	
baseUrl	https://api.codes.lacounty-isab.org	https://api.codes.lacounty-isab.org	

Exercise 3 (conclusion)

20. We can see from the environment view that we're missing the API key. Rather than navigating back to the environment section to edit the current environment, just click the **Edit** link in the viewer.
21. Add the API key like before, using the production value instead of the test value.
 - a. **Variable:** apikey
 - b. **Initial Value:** Your PROD API key
 - c. **Current Value:** <PROD API key>
22. Save the environment.
23. Return to the *Get charge 826* tab and send the request. This should return the production result since the **Production** environment should still be selected.
24. Select the **Test** environment and re-run.
25. Select *No Environment* and re-run.

Postman environments are very important for

- keeping your workspace organized by reducing the need to duplicate requests for different environments
- tracking which values are environment-specific
- limiting the visibility of sensitive environment variables.

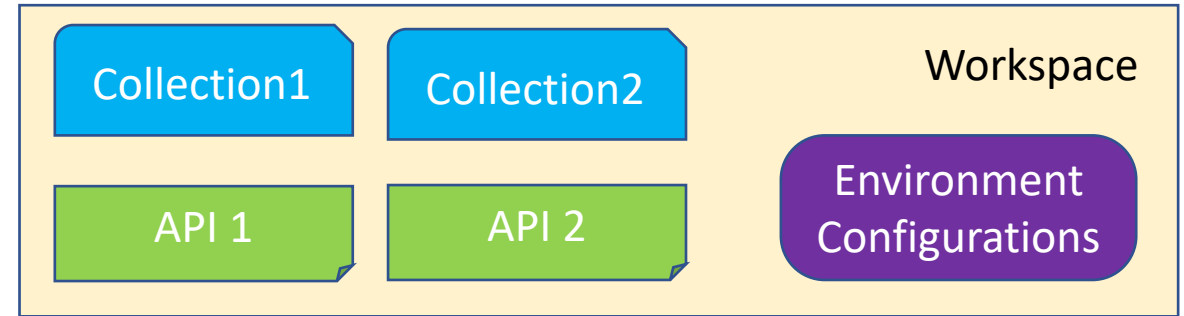
This last point is not as apparent for a **private** workspace. In team workspaces it allows developers to keep their keys, tokens and passwords confidential.

A set of environment definitions is scoped to a workspace. Each workspace has its own set of environment definitions.

Test		Your team sees this	Only you can see this
	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	baseUrl	https://api-test.codes.lacounty-isab.org	https://api-test.codes.lacounty-isab.org
<input checked="" type="checkbox"/>	apikey	Your API key	[REDACTED]

Concept Review

- **Workspace** – a unit of access control
 - *Personal* workspace – Artifacts to which only you have access.
 - *Private* workspace – Only invited team members can access
 - *Team* workspace – Artifacts accessed by members of the workspace.
 - Often a team workspace starts as personal until ready to be shared; then made private or team by inviting team members.
- **Collection** – a set of request definitions
 - Can be subdivided into folders
 - Configuration variables, scripts, and authentication methods can be set at the collection, folder or request level.
 - It's common to dedicate a collection to an API specification.
 - A collection is defined inside a workspace. It may be copied, moved or synchronized between workspaces.



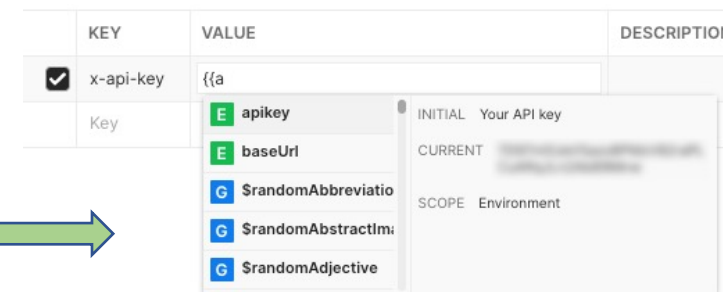
- **Environments** – Variables defined with values particular to each environment. Variables are valid throughout the workspace.
- **API** – An OpenAPI specification (either in YAML or JSON format).
 - Provides the interface specification for your API
 - Payload formats
 - Operation methods and URIs
 - Header requirements
 - Security requirements
 - Expected response codes and their associated formats.
 - An API can be the basis for generating a collection.
 - An API is defined inside a workspace; usually the same workspace in which its associated collections are defined.

Exercise 4 – Request Headers

1. Change to the **Collections** view and choose the **CJIS Tables** folder. This is the folder in which we configured our API Key authorization.
2. Change the **Type** of the authorization back to *Inherit auth from parent* and save the change.
3. From the environment dropdown, select **Test**.
4. Go back to the *Get charge 826* request and click **Send** to verify that our API key is no longer sent.
5. Select the **Headers** tab of the request definition. This should present a table of header values.
6. Add the API key to the header.
 - a. **Key:** x-api-key
 - b. **Value:** {{apikey}}
 - c. **Description:** API key for CJIS Tables
7. Try the request again to verify API key explicitly added to the header works the same way as when configured through the authorization tab.

Whether we configure through the **Authorization** section or as a **Header** depends on the context.

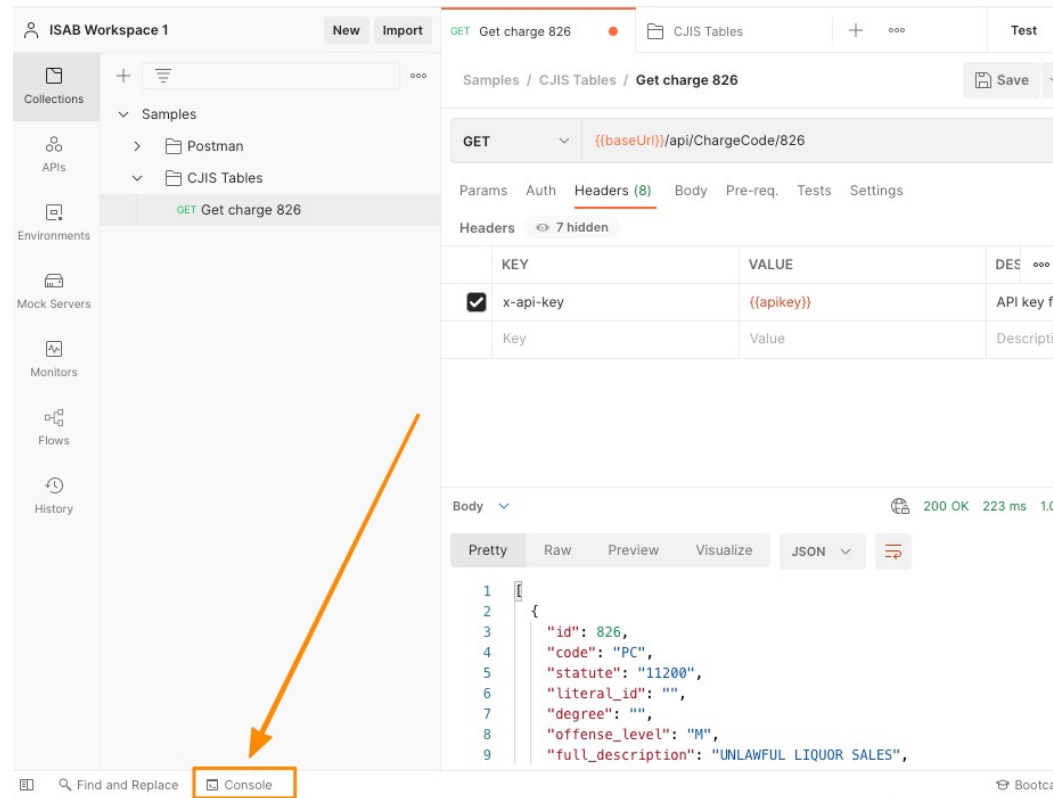
- The Authorization section provides scope options.
- The Header gives more fine-grained control. It's the right option to use when an API key is only used for throttling purposes and a different method is used for user authorization.



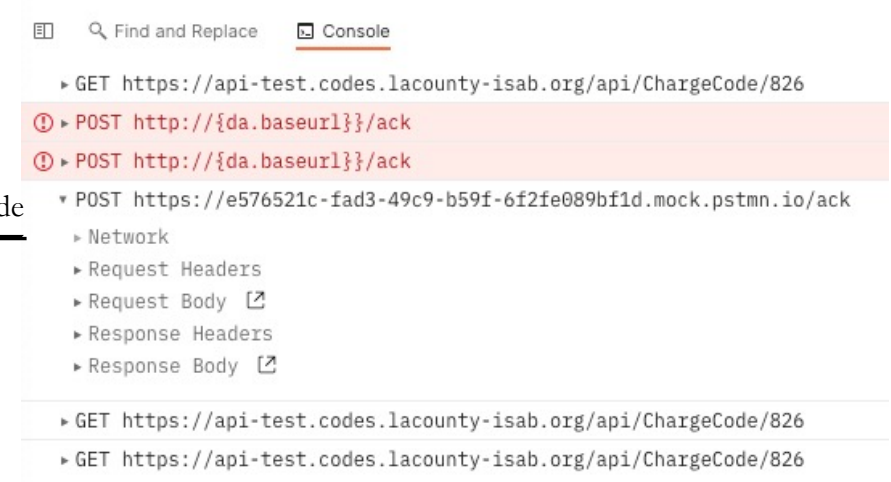
The UI provides auto-complete help that displays existing environment entries.

Request Details – The Console

A good amount of information about the requests and response are available within their respective panes in the UI. Many more details are available through the **Console** link.



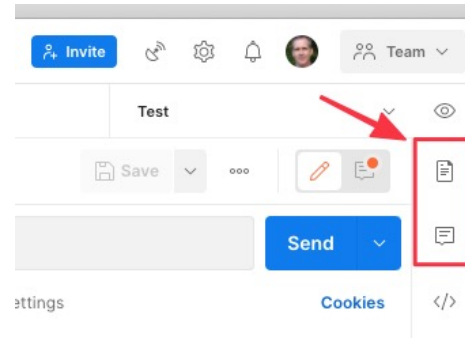
Expand the
Network node



The console provides detailed information about the request, response, network protocols and SSL certificate details.

Exercise 5 – Documentation and Comments

On the right side of the request definition, there are buttons for adding documentation and comments to a request.



1. Select the Postman Echo request we created earlier.
2. Click the **Documentation** button.
3. Click the pencil icon to active the Markdown edit box.
4. Type "This is just for getting ****started****." in the box and Save.
5. Click the **Comments** button.

6. Click **Add Comment** and type something like

That's nice; but how about invoking an API of your own.

7. Click the **Add Comment** button at the bottom.

The value of documentation and comments will become more apparent in a **team workspace** setting. It keeps the conversation about the request local to the request; rather than dispersed across several email threads.

Team Workspaces

- Motivation
 - Avoids export/import of requests
 - Reduces errors
 - Keeps API specification, documentation and requests together.
 - Reduces email clutter from
 - workspace setup
 - defining new requests
 - API schema changes
- Next exercise will
 - simulate a team workspace
 - can't actually share a workspace due to limited licenses
 - most private/team features are similar
 - working with multiple workspaces is the main difference
 - obtain update credentials
 - issue login API call
 - store token using script
 - reference the token for update API
 - use dynamic variables

Exercise 6 – POST Request

In this exercise we'll be using *temporary* credentials that will only be valid during participation in the delivery of the workshop.

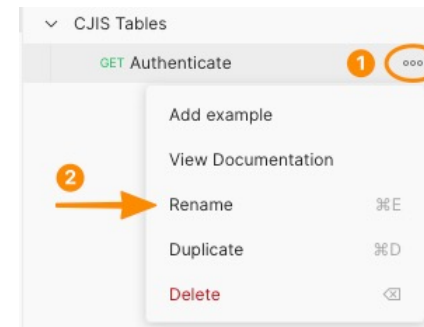
1. From the **Workspaces** dropdown menu click **Create Workspace**.
 - a. **Name:** ISAB Workshop 2
 - b. **Summary:** Used to demonstrate scripts, dynamic variables, and other HTTP methods
 - c. **Visibility:** Personal
 - d. Click **Create Workspace**.
2. Select **Collections** from the navbar if it isn't selected already. Since this is a new workspace, there should be zero collections. Click the **Create collection** button.
3. Name the new collection **CJIS Tables**. For this exercise, we'll dispense with the folders and create the requests directly under the collection.
4. Click **Add a request** and name it **Authenticate** by clicking the pencil icon next to the name. The request can also be renamed by selecting its triple dots and choosing **Rename**.
5. Change the method to **POST**.

6. Set the request URL:
`https://auth-test.codes.lacounty-isab.org/idp/login`
7. Select **Headers** for the request and add the 'TEST' API key as the `x-api-key` request header.
8. Select **Body** for the request and choose **raw** as the format. Enter the following JSON.

The `<userid>` and `<passwd>` values should come from the MS Teams channel.

```
{
  "id": "<userid>",
  "pw": "<passwd>"
}
```

9. Send the request. The **Body** of the **Response** section should be a JSON object with a `token` property containing an encoded JWT.
10. Save the request.
11. (Optional) Paste the encoded token value into the debugger at `https://jwt.io` to see the decoded token.



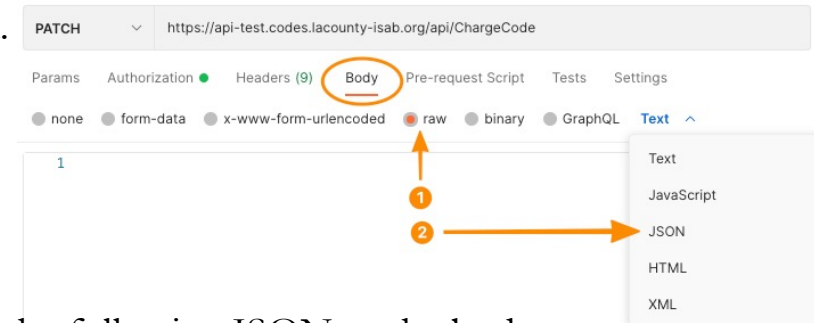
Exercise 7 – PATCH Request

The HTTP PATCH method is used to update an existing record. In this request, we'll use the token from the previous exercise to update the short description of a record in the TEST environment.

1. Create a new request in the **CJIS Tables** collection
 - a. **Name:** Change short description
 - b. **Method:** PATCH
 - c. **URL:** `https://api-test.codes.lacounty-isab.org/api/ChargeCode`
 - d. Save the request.
2. Select the **Authorization** tab of the request. Notice it's set to inherit its configuration from its parent (the collection level). By configuring the authorization at the collection level, the authorization would be available to all requests. For this small example, we'll configure at the request level.
3. For **Type**, select **Bearer Token**. The right side should display a **Token** field in which to include a token. We will fill this in once we know the value of the token.
4. Add the `x-api-key` header with the TEST API key.

Note: Using the Header section for the API key allowed us to use the Authorization section for the bearer token.

5. Select the **Body** tab. For the format, select **raw** and then select JSON from the **Text** dropdown as shown on the right.



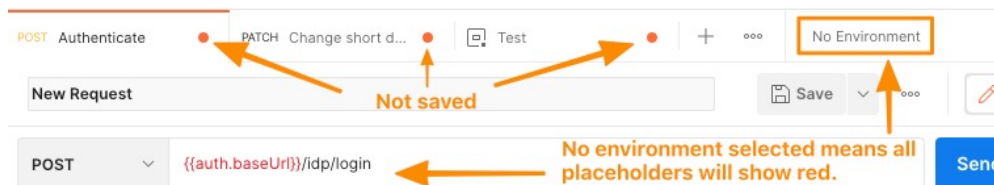
6. Add the following JSON to the body.
- ```
{
 "id": "826",
 "short_description": "UNLAWFUL LIQUOR 2"
}
```
7. Verify that the request returns 401 without a valid bearer token.
  8. Go back to your POST Authenticate request and copy the token value from the response. Invoke it again if the token is old or missing.
  9. Go back to the PATCH request and paste the token value into the bearer token value of the Authorization tab.
  10. Resend the PATCH request. A successful update should return the following:

```
{
 "changedRows": 1
}
```

# Exercise 8 – Create Environment

At this point all our values are pinned to the 'TEST' environment. This is a good time to create an environment entry for TEST.

1. Select the POST Authenticate request.
2. Copy the base URL, the part starting with "https" and ending with ".org".
3. Click the **Environments** entry of the left navbar.
4. Click **Create Environment**.
5. Name the environment **Test**.
6. Enter the following for the first entry:
  - a. **Variable:** auth.baseUrl
  - b. **Initial Value:** https://auth-test.codes.lacounty-isab.org
7. Switch back to the POST Authenticate request.
8. Substitute the base URL value in the URL field with the variable {{auth.baseUrl}}.



9. Save changes to requests and environment.
10. Activate the Test environment. The environment variable text within the request should turn orange.
11. Copy the value of the x-api-key header.
12. Create a new Test environment entry:
  - a. **Variable:** apikey
  - b. **Initial Value:** Your API key
  - c. **Current Value:** <Test API key>Note the initial and current values are different.
13. Repeat the same process for the **id** and **pw** values for the payload. The variable names should be user.id and user.pw, respectively.
14. Add the api.baseUrl to the environment with an initial value of https://api-test.codes.lacounty-isab.org.
15. Save the Environment.
16. Substitute the variables into the environment body of the POST request.
17. For the PATCH request, add the api.baseUrl to the URL field.
18. Add the existing {{apikey}} to the PATCH header section.
19. Invoke the request to verify it still works.

This is how your environment should look when done. Only the first two rows have explicit initial values.

|                                     | VARIABLE     | INITIAL VALUE ⓘ                           | CURRENT VALUE ⓘ                           |
|-------------------------------------|--------------|-------------------------------------------|-------------------------------------------|
| <input checked="" type="checkbox"/> | auth.baseUrl | https://auth-test.codes.lacounty-isab.org | https://auth-test.codes.lacounty-isab.org |
| <input checked="" type="checkbox"/> | api.baseUrl  | https://api-test.codes.lacounty-isab.org  | https://api-test.codes.lacounty-isab.org  |
| <input checked="" type="checkbox"/> | apikey       | Your API key                              |                                           |
| <input checked="" type="checkbox"/> | user.id      | User ID                                   |                                           |
| <input checked="" type="checkbox"/> | user.pw      | User Password                             |                                           |

# Exercise 9 – Collection Variables and Scripts

We are now able to invoke an API requiring an authentication token by invoking the token service and then the update service. But it still requires we *manually* copy and paste the token from the response body of one service into the authorization configuration of another. In this exercise, we'll script the copy-and-paste.

1. Select the POST Authenticate request.
2. Select the **Tests** tab. A JavaScript code editor is displayed. Code in this tab runs **after** the request completes. Enter the following code into the editor.

```
var jsonData = pm.response.json();
var token = jsonData.token;
if (pm.response.code === 200) {
 if (token) {
 pm.environment.set("authHeader", token);
 console.log('Token stored in authHeader variable');
 } else {
 console.log('token response is null');
 }
} else {
 console.log('reponse code not 200');
}
```

If the authentication returns a valid token, the above script will set a Postman variable named `authHeader` in the **environment** scope.

3. Select the PATCH request and select the **Authorization** tab. Replace the **Token** value with `{{authHeader}}`. Save both requests.

4. Run the Authenticate request.
5. Run the Update request.
6. Show the console.

Notes on the script code, reproduced below with line numbers and color highlighting:

- The **pm** variable is initialized by Postman before the script runs. It is an object with several references for accessing details about the environment, the request and the response.
- Line 1 retrieves the JSON from the response.
- Line 2 stores the token.
- Line 5 stores the token in the **environment** scope.

For reference, search Google for "Postman sandbox API reference".

```
1 var jsonData = pm.response.json();
2 var token = jsonData.token;
3 if (pm.response.code === 200) {
4 if (token) {
5 pm.environment.set("authHeader", token);
6 console.log('Token stored in authHeader variable');
7 } else {
8 console.log('token response is null');
9 }
10 } else {
11 console.log('reponse code not 200');
12 }
```

# Exercise 10 – Dynamic Variables

The Update response from the last exercise might have returned

```
{"changedRows": 0}
```

That's because we used the same value for the first update. We only see a change if we use a different value. But that requires manually changing the PATCH request body.

This is a common desire: to send the same structure of a request, but with different values of the same type each time. This is supported in Postman through *dynamic variables*.

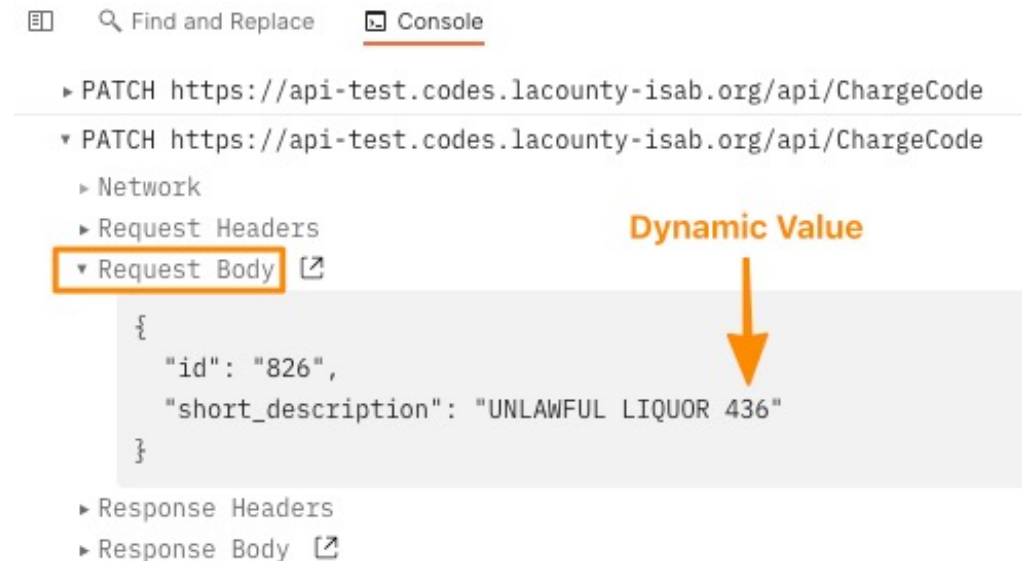
1. Navigate to the PATCH request body.
2. The value of the `short_description` field should still read "UNLAWFUL LIQUOR 2". Change this to "UNLAWFUL LIQUOR {{\$randomInt}}".
3. Save the request and send again. It should show that the number of rows changed is 1. But what did it change to?
4. Expand
  - the **Console**
  - the last request
  - the Request Body

This should reveal the request that was sent.

Postman supports\* many such variables such as:

- names, titles, addresses and emails
- dates and times
- GUIDs, IPs and filenames

Google "Postman dynamic variables" for the full list.



\* Postman support for dynamic values comes from the Faker JavaScript library.  
<https://rawgit.com/Marak/faker.js/master/examples/browser/index.html>



# Appendix: Bluecoat Root Certificate

Early in the workshop we discussed why you might need to use the web application version of Postman if you're within an LA County internal network. The network perimeter security interferes with the native application's ability to "phone home."

Another problem that may present itself is the invocation of remote APIs over HTTPS. The LA County network perimeter will terminate the connection and swap SSL server certificates with one signed by its root. But a default Postman installation will not recognize this root. The symptom of this problem is a warning in the console:

**Warning: Unable to get local issuer certificate**

The request may still succeed, but your API client may not be so lucky if the trust store is not properly configured.

You can remove this warning by configuring the LA County perimeter root issuer within the Postman certificate store. This process is described in this and the next slide.

The first step is to acquire a copy of the perimeter issuer certificate. A copy is attached below if you don't have one handy. Just copy the contents in the grey box below to a file named `lac-sof-root.pem`. A quick check can verify the subject.

```
openssl x509 -in lac-sof-root.pem -noout -subject
subject=DC = COM, DC = LAC, CN = LAC-SOF-Root CA II
```

```
-----BEGIN CERTIFICATE-----
MIIDaTCCAIGgAwIBAgIQLA+VADnZMZFE8sdynYppIDANBgkqhkiG9w0BAQsFADBH
MRMwEQYKCZImiZPyLGQBGRYDQ09NMRMwEQYKCZImiZPyLGQBGRYDTEFDMRswGQYD
VQQDExJMQUmtU09GLVJvb3QgQ0EgSUkwHhcNMjYxMTIxMTgyMjU5WhcNMjYxMTIx
MTgzMjU5WjBHRMwEQYKCZImiZPyLGQBGRYDQ09NMRMwEQYKCZImiZPyLGQBGRYD
TEFDMRswGQYDVQQDExJMQUmtU09GLVJvb3QgQ0EgSUkwggEiMA0GCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQPdpF3yIhdS7R4ElzXmqzh00SdIaXOp+GCG+gEmFk5Q
DrdDTlzDlOQxiM+dDv49QEH+hEUSWT2ROfpXWhTp6ZO7I9M9lveCARGaMM5Zn4oX
unG+ATDYvMQpt6n/VtHCgoY4d+KeqYPJqm8yCrLGUhlOmLJ8ZD9EQ+dcHvJKUAHC
Ugd+CsEwQ9kd/lJpaV9dKZ88ZReIw5yJorWbNfqrgrW+AnuGpJWQEZWbZHxjl42j
pAl1R99m3/szkCO3rN0jATt8W0ylM9XqGuaGCB6021cGItsSl6k93A1qRf5Ms+Se
nLtrbrwcu9/rrqsESx55wc4ArRVUMBvHAreOFcMsa3SchAgMBAAGjUTBPMAsGA1Ud
DwQEAwIBhjAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBSIy8+g/VdchH65hEP7
N6gIQpKw5DAQBgkrBgEEAYI3FQEEAwIBADANBgkqhkiG9w0BAQsFAAOCAQEAv2H
3OX9LravtVGul7QiZm7hUCHMYC4Jdl19PAY0wDL10KDhoNRNhdVwjUs1As04pKL
NZQozRn8KoMjHE1TpoFN0hiLOMbKetNfruyLCTUpEI5+axBBOyAzvm0yo0MmuXw
cmT5/lS0FH2hos0AfrHO/ralowAxrTWEdu/8ZOvJRXmvyYBOxfRQOR2kSdhe3Aw
pnzx54vRaoyeCMLnxKQOD87/Y+aqA/A08KAeYKbFBlkNdBE09H8av/2LK7gs4Doj
ZCcHUueOmfqnVFimkDbA6oVZgw1HE5s94LZFW8prCgGyaBs6buocQkkyhq1E0ob/
StZ7ql7oBR1gIFHE9g==
-----END CERTIFICATE-----
```

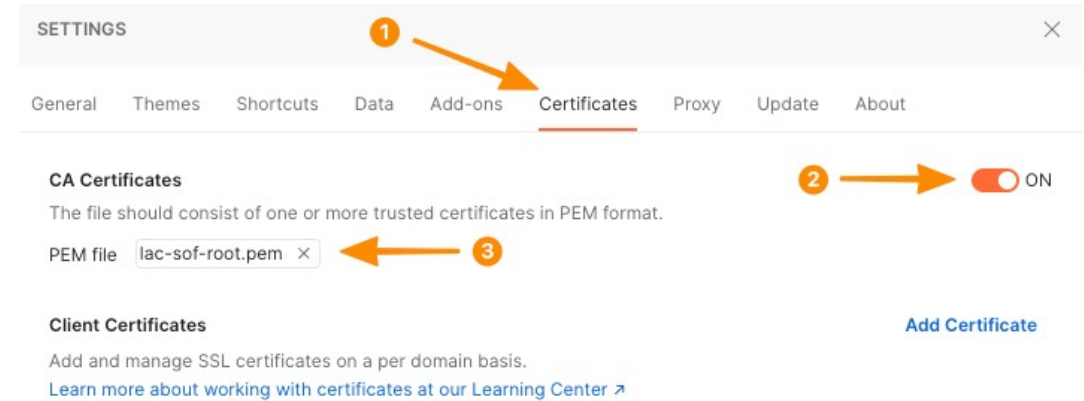


# Appendix: Install Issuer

To install the LA County issuer certificate as a CA (Certificate Authority), open the Postman **Preferences** dialog. It should appear as shown in the screen shot.

1. Select the **Certificates** tab.
2. Enable CA Certificates by toggling the setting to **ON**.
3. Load the CA certificate file from the previous slide.

Dismiss the dialog and try again. The issuer warning should disappear on your next request.



**Note:** If you try to invoke an SSL server **within** your local network (with no intermediate SSL termination), this will cause the problem again. Simply disable the toggle in Step 2 above to return to the default certificate store.

# Appendix: Postman Part 2 ?

This workshop has focused on working with **requests**, which is the bread-and-butter of Postman. Another important aspect of a Postman workspace is the **API specification**. This allows us to share what the API implementation expects of the clients. That opens the potential for a "Part 2" of this workshop focused on working with API specifications. Potential topics include

- The OpenAPI 3.0 Specification
  - YAML formats
  - JSON schema formats
  - Mixing the two
- Adding an OpenAPI schema to Postman.
- Mock Services

This doesn't seem like much, but the OpenAPI specification could be a workshop on its own.

