Postman

API Development Collaboration

- Part 2 Agenda
  - Other HTTP Methods
  - Authentication Tokens
  - Storing credentials
  - Variable Scopes
  - Dynamic Variables
  - Simple Scripts

- Assumption – you are comfortable with material from Part 1.

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

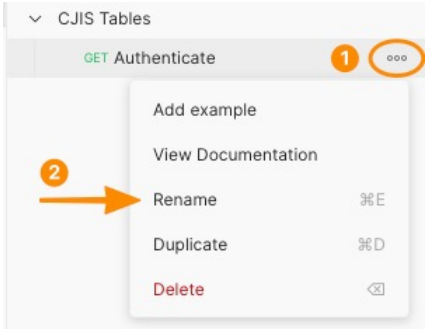
In this exercise we'll be using *temporary* credentials that will only be valid during participation in the delivery of the workshop.

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.



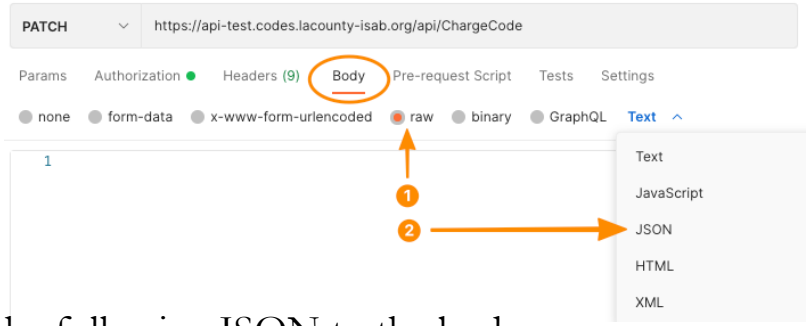
Exercise 7 – PATCH Request

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**: 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"
}
```

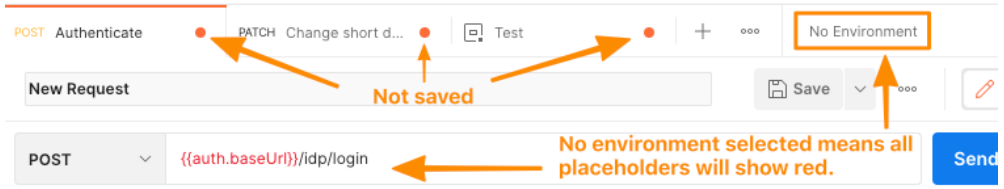
7. Verify that the request returns 401 without a valid bearer token.
8. Go back to your POST Authenticate request and copy the token value from the response. Invoke the login again if the token is old or missing.
9. Go back to the PATCH request and paste the token value into the bearer token value of 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" to your clipboard.
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` (paste from clipboard)
   c. **Current Value**: Same as initial value
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 the environment and the requests.



10. Activate the **Test** environment. The environment variable text within the request should turn from red to orange.
11. Copy the value of the `x-api-key` header to the clipboard.
12. Create a new Test environment entry:
    a. **Variable**: apikey
    b. **Initial Value**: Your API key
    c. **Current Value**: <Test API key> (from clipboard)

    Note the initial and current values are different.
13. Repeat the same process for the **id** and **pw** values for the body. The variable names should be `user.id` and `user.pw`, as shown below.
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. For the POST request, substitute the variables into the environment body.
17. In the header section substitute `{{apikey}}` for the value of `x-api-key`.
18. For the PATCH request, add the `api.baseUrl` to the URL field.
19. Add the existing `{{apikey}}` to the PATCH header section.
20. 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
☑	auth.baseUrl	https://auth-test.codes.lacounty-isab.org	https://auth-test.codes.lacounty-isab.org
☑	api.baseUrl	https://api-test.codes.lacounty-isab.org	https://api-test.codes.lacounty-isab.org
☑	apikey	Your API key	
☑	user.id	User ID	
☑	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 PATCH 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".

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 need: 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*.

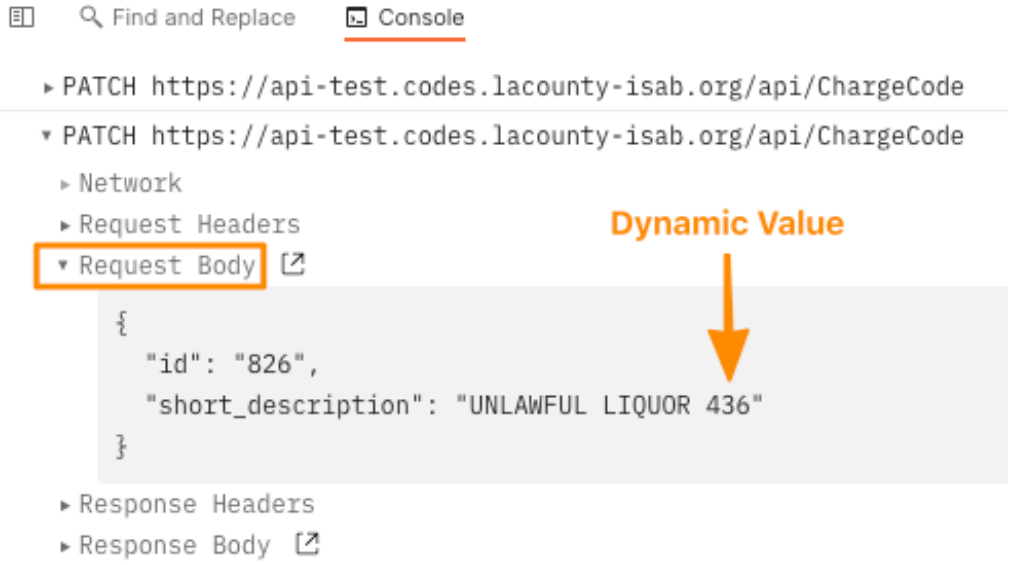
1. Navigate to the PATCH request body.
2. The value of the `short_description` field should still read `"UNLAWFUL LIQUOR 2"`. Change this to `"UNLAWFUL LIQUOR {{$randomInt}}"`.
3. Save the request and send again. It should show that the number of rows changed is 1. But what did it change to?
4. Expand
   - 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-----
MIIDaTCCAlGgAwIBAgIQLA+VADnZMZFE8sdynYppIDANBgkqhkiG9w0BAQsFADBH
MRMwEQYKCZImiZPyLGQBGRYDQ09NMRMwEQYKCZImiZPyLGQBGRYDTEFDMRswGQYD
VQQDExJMQUMtU09GLVJvb3QgQ0EgSUkwHhcNMTYxMTIxMTgyMjU5WhcNMjYxMTIx
MTgzMjU5WjBHMRMwEQYKCZImiZPyLGQBGRYDQ09NMRMwEQYKCZImiZPyLGQBGRYD
TEFDMRswGQYDVQQDExJMQUMtU09GLVJvb3QgQ0EgSUkwggEiMA0GCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQDpoF3yIhdS7R4E1zXmqzhO0SdIaXOp+GCG+gEmFk5Q
DrdDTlzD1OQxiM+dDv49QEH+hEUSWT2ROfpXWhTp6ZO7I9M9lveCARGaMM5Zn4oX
unG+ATDYvMQpt6n/VtHCgoY4d+KeqYPJqm8yCrLGUhlOmLJ8ZD9EQ+dcHvJKUAHC
Ugd+CsEwQ9kd/lJpaV9dKZ88ZReIw5yJorWbNfqrgRW+AnuGpJWQEZWBZHxjl42j
pAl1R99m3/szkCO3rNOjATt8WOy1M9XqGuaGCB6O21cGItsSl6k93A1qRf5Ms+Se
nLtbrwcu9/rrqsESx55wc4ArRVUMBvHAreOFcMsa3SchAgMBAAGjUTBPMAsGA1Ud
DwQEAwIBhjAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBSIy8+g/VdchH65hEP7
N6gIQpKw5DAQBgkrBgEEAYI3FQEEAwIBADANBgkqhkiG9w0BAQsFAAOCAQEAxv2H
3OX9LravtVGul7QiZm7hUChMYC4Jdl19PAY0wDLl0KDhoNRNhdVwjrUs1As04pKL
NZQozRn8KoMjHE1TpoFN0hiLOMbKetNfruuyLCTUpEI5+axBBOyAzvm0yo0MmuXw
cmT5/ls0FH2hos0AfrHO/ra1owAxrTWEdu/8ZOvJRXmvyYBOxfRQOR2kSdhe3AWr
pnzx54vRaoyeCMLnxKQOD87/Y+aqA/A08KAeYKbFBlkNdBE09H8av/2LK7gs4Doj
ZCcHUueOmfqnVFimkDbA6oVZgw1HE5s94LZFW8prCgGyaBs6buocQkkyhq1E0ob/
StZ7ql7oBR1gIFHE9g==
-----END CERTIFICATE-----
```

Appendix: Install Issuer

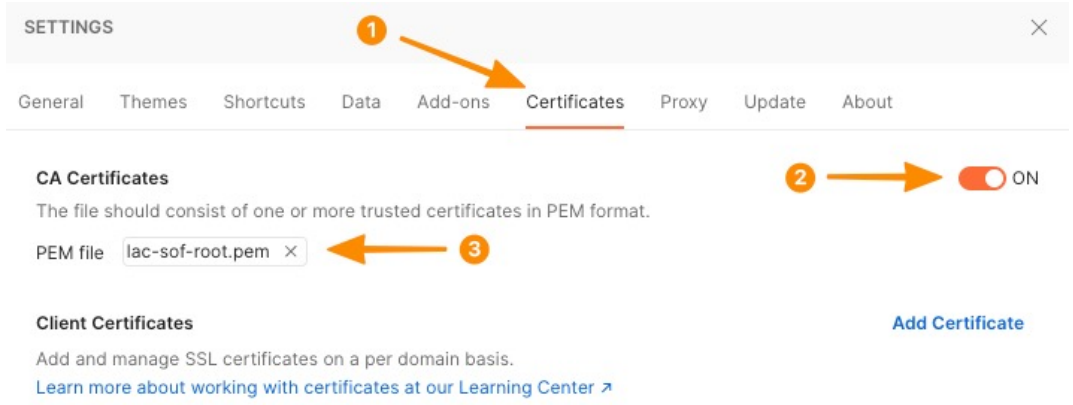
To install the LA County issuer certificate as a CA (Certificate Authority), open the Postman **Preferences** dialog. It should appear as shown in the screen shot.

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 3 ?

This workshop has focused on working with other HTTP methods besides GET and how to populate variables with values that can be reused across other requests.

Part 3 will demonstrate more tricks we can employ using scripts to

- gain more control of random values for requests
  - numbers of a fixed size
  - choose from a fixed set of choices
  - pick dates in the future
- process XML results