



Lab Guide

ServiceNow custom REST APIs:
Build Custom Services the right
way with Scripted REST APIs

This
Page
Intentionally
Left
Blank

Lab Goal

Before we get started building custom services with Scripted REST APIs, we need to get our lab instance setup. In this lab you will be modifying an existing scoped application. Start out by importing the Polls Application from Source Control. Follow the directions below to fork this application to your GitHub account and begin working.

Lab Setup

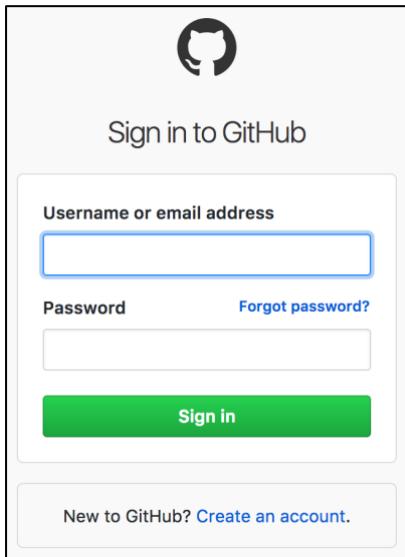
Prerequisites

In order to complete this lab, you must:

- Create a GitHub account, if you do not already have one.
- Install Postman from <https://getpostman.com> if you do not already have it.

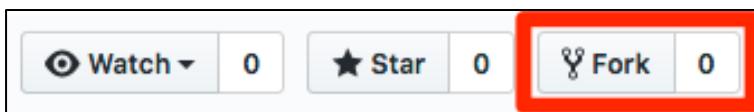
Fork the Lab GitHub Repository

1. Log in to your GitHub account at <https://github.com/login>.



2. Navigate to: <https://github.com/balazsburgermeister/ScriptedRESTAPI>

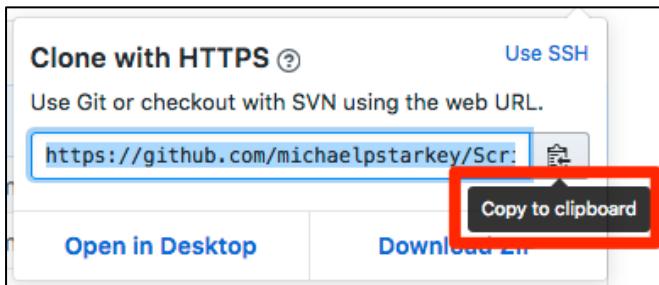
3. Click **Fork**.



4. Note in the upper left that the repository has been copied to your account. You now have a copy of the lab material for reference after the conference!

5. Locate and click on the **Clone or download** button and then click the clipboard to the right. This action copies the URL in the clipboard.

IMPORTANT: Be sure to copy the **HTTPS** repo URL in GitHub.

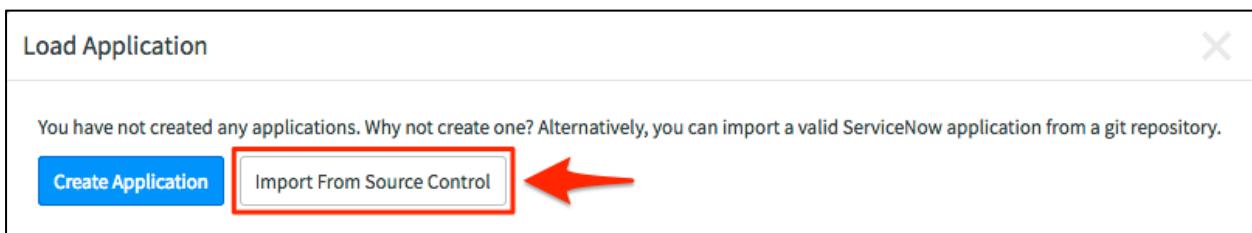


Import the Polls Application from Source Control

6. Log in to your instance with the credentials provided on the cover sheet of this document.
7. Navigate to **System Applications > Studio**.



8. Click **Import From Source Control**.



9. In the Import Application window, paste the URL copied in step 5 and provide your GitHub credentials. Click **Import**.

The screenshot shows the 'Import Application' dialog box. It contains a descriptive text about importing from source control, a 'URL' field with the value 'https://github.com/michaelpstarkey/ScriptedRESTAPI.git', and fields for 'User name' and 'Password'. At the bottom are 'Cancel' and 'Import' buttons.

10. When the import completes, click **Select Application**.

The screenshot shows a success message: 'Successfully applied commit 299d811fd0e5a275d5939f0c64aa4d2f9e1ce970 from source control'. A 'Select Application' button is visible at the bottom right.

11. Click on the **Polls** application you just imported.

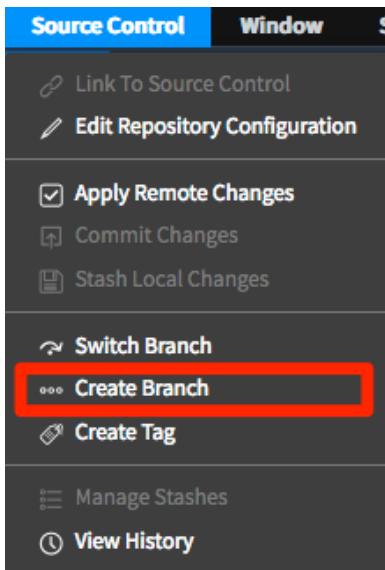
The screenshot shows the 'Load Application' dialog box. It has tabs for 'Create A New Application' and 'Import From Source Control'. Below is a table titled 'Applications (1)' with one row for 'Polls'. A red arrow points to the 'Polls' link in the table. The table columns are: Status, Application, Vendor, Version, Created on, and Updated On.

Status	Application	Vendor	Version	Created on	Updated On ↓
	Polls		1.0.0	2016-04-18	2016-04-18 18:22:25

You've now successfully imported your forked version of the application for use in this workshop.

Get ready for Lab 1 – Create a new branch from Lab1-start tag in Studio

12. In Studio, navigate to Source Control > Create Branch.



13. In the pop-up window, enter a branch name, then select **Lab1-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab1-branch**

Create from Tag: **Lab1-start**

The dialog box has a title 'Create Branch' and a message: 'Creating a branch will result in a new branch being created in the remote repository that is configured for this application.' It contains two input fields: 'Branch Name' (containing 'my-Lab1-branch') and 'Create from Tag' (containing 'Lab1-start'). A 'Cancel' button and a 'Create Branch' button are at the bottom. The 'Create from Tag' dropdown is highlighted with a red box.

14. When the create is complete, click **Close Dialog** in the Create Branch pop-up.

15. Verify Studio is on branch **my-Lab1-branch** from the bottom right corner of the screen.

Polls | 1.0.0 0 Files (0 unsaved)

my-Lab1-branch

Lab setup is complete. You are now ready to start Lab 1.

Lab Goal

The purpose of this lab is to familiarize yourself with ServiceNow Scripted REST APIs. In this first lab you'll build a Scripted REST API that returns "Hello, world!" in response to a GET request. After building the API you'll use the ServiceNow REST API Explorer and API testing tool Postman to make requests to the REST API.

Prerequisite

- Knowledge of REST APIs
- Knowledge of HTTP clients
- Postman API testing tool. To get Postman go to: <https://www.getpostman.com/>

Lab 1
Build
“Hello,
world!”

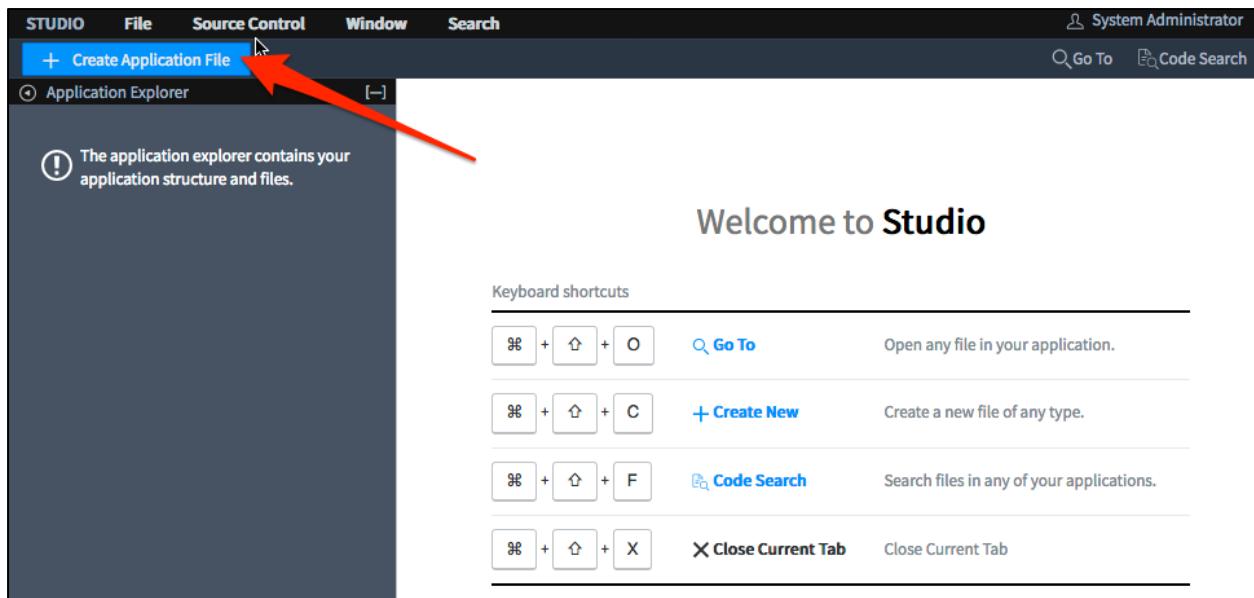
Create Lab 1 starting branch

1. If you completed the lab setup, proceed to the next step.

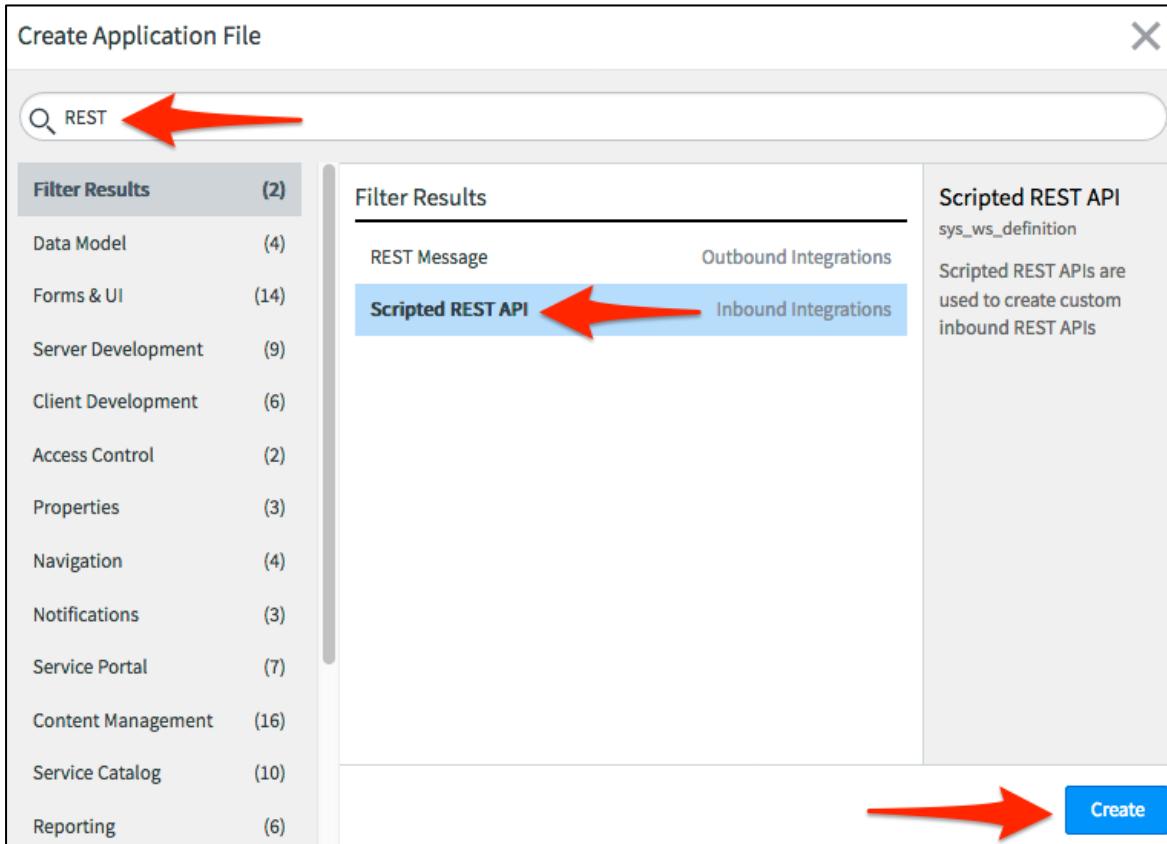
If you haven't yet completed lab setup, follow the steps in lab setup to create the **my-Lab1-branch** from the **Lab1-start** git tag.

Create the Hello World Scripted REST API

2. In Studio, click **Create Application File**.



3. In the **Create Application File** window, type **REST** in the filter then select **Scripted REST API** and click **Create**.



4. Give the API a name. Note the API ID populates automatically from the API Name, but can be changed.

Name: **Hello, world!**

The screenshot shows the 'Scripted REST Service' configuration page. A red box highlights the 'Name' field containing 'Hello, world!', which has a red arrow pointing to it. Another red arrow points to the 'Submit' button at the bottom left. The 'API ID' field contains 'hello_world'. The 'Application' field is set to 'Polls'. The 'API namespace' field is set to 'x_snc_polls'. The 'Protection policy' dropdown is set to '-- None --'.

Click **Submit**.

5. Add a **Resource** to the API by finding the **Resources** related list. Click **New**.

The screenshot shows the 'Resources' related list. A red box highlights the 'Resources' tab, and another red arrow points to the 'New' button, which is highlighted with a blue background. The list header includes filters for 'Name' and 'Search'. Below the header, a note says 'API definition = Hello, world!'. The list is currently empty, displaying 'No records to display'.

6. Specify the following properties for the new resource and complete the script.

Name: **Hello resource**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab1

The screenshot shows the 'Scripted REST Resource' configuration page for a resource named 'Hello resource'. The 'Name' field is highlighted with a red box and an arrow pointing to it. The 'Script' section contains the following code:

```
1 (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     return "Hello, world!";  
3 })(request, response);
```

Click **Submit**.

Test with REST API Explorer

7. Click **Explore REST API**.

The screenshot shows the 'Related Links' section of the ServiceNow interface. The 'Explore REST API' link is highlighted with a red arrow. Below it, there's a table listing a single resource named 'Hello resource' with a GET method and the path '/api/x_snc_polls/hello_world'. The table has columns for Name, HTTP method, Relative path, and Resource path.

Name	HTTP method	Relative path	Resource path
Hello resource	GET	/	/api/x_snc_polls/hello_world

8. The **REST API Explorer** opens in a new browser window. Click **Explore**.

The screenshot shows the 'REST API Explorer' interface. It features a lightbulb icon and the title 'REST API Explorer'. A descriptive text explains that the explorer allows quick access to the ServiceNow inbound REST API. At the bottom, there is a blue 'Explore' button, which is highlighted with a red arrow.

9. The “Hello, world!” Scripted REST API is pre-selected in the Explorer menus. Click **Send**.

The screenshot shows the REST API Explorer interface. On the left, there are three dropdown menus: 'Namespace' (set to 'x_snc_polls'), 'API Name' (set to 'Hello, world!'), and 'API Version' (set to 'latest'). Below these is a link 'Hello resource (GET)'. The main area displays the 'Hello, world!' API details, which are also highlighted with a red box. The 'Hello resource' section contains a 'GET' method with the URL '/api/x_snc_polls/hello_world'. Below this is a 'Prepare request' section with 'Query parameters' and 'Request headers' tables. The 'Request headers' table has three rows: 'Request format' (Value: application/json), 'Response format' (Value: application/json), and 'Authorization' (Value: Send as me). At the bottom left of the main area, there is a blue 'Send' button. A red arrow points to this 'Send' button. At the bottom right, there is a row of links: [ServiceNow Script] [cURL] [Python] [Ruby] [JavaScript] [Perl] [Powershell].

10. Verify response status code is **200 OK** and response body is “Hello, world!”.

Request	
HTTP Method / URI	GET /api/x_snc_polls/hello_world
Headers	
Accept	application/json
X-UserToken	cbc451c1f51232007f4494386d91c994797a9010e7105820322bb160a4a94b657e7dcd0c
Response	
Status code	200 OK 
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Fri, 21 Apr 2017 11:45:21 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-encoding	chunked
X-is-logged-in	true
Response Body	
{ "result": "Hello, world!" }	

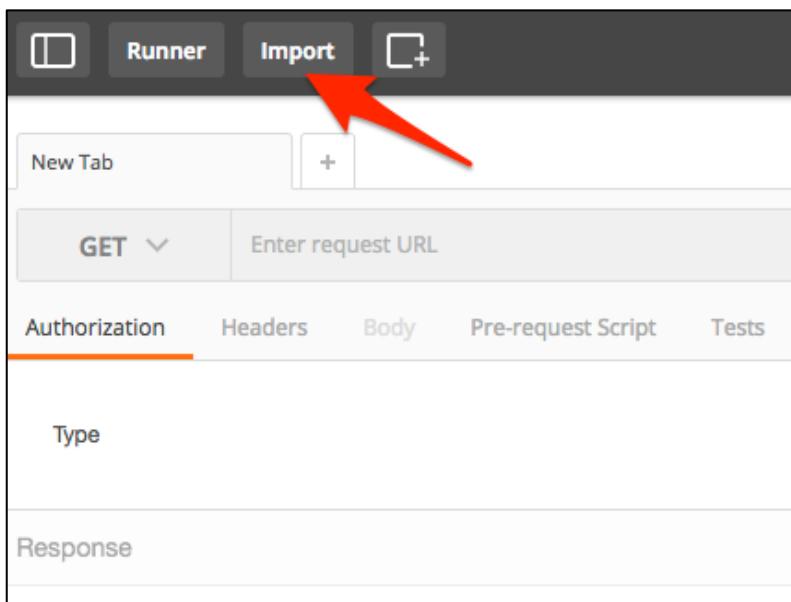
Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

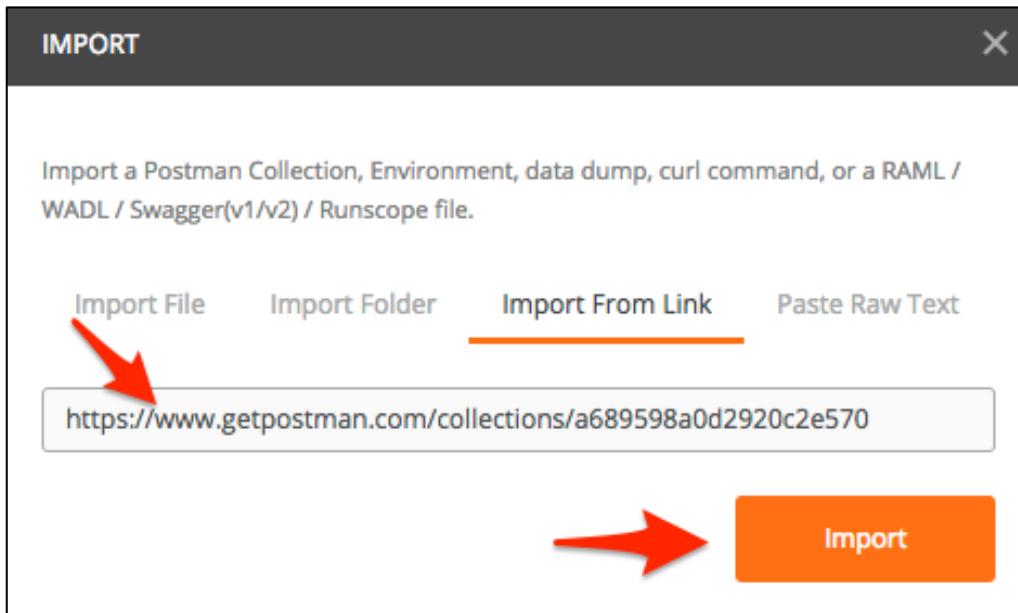
11. Similar to creating the Lab1 starting branch, the completed lab can also be checked out from a tag (**Lab1-complete**) in Source control.
12. In **Studio**, navigate to **Source Control > Create Branch**.
13. In the pop-up window, enter a branch name, then select **Lab1-complete** from the **Create from Tag menu**, and click **Create Branch**.
Branch: **my-Lab1-branch-complete**
Create from Tag: **Lab1-complete**
14. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
15. Verify Studio is on branch **my-Lab1-branch-complete**.
16. You are now ready to continue with the next section of **Lab 1**.

Test with Postman

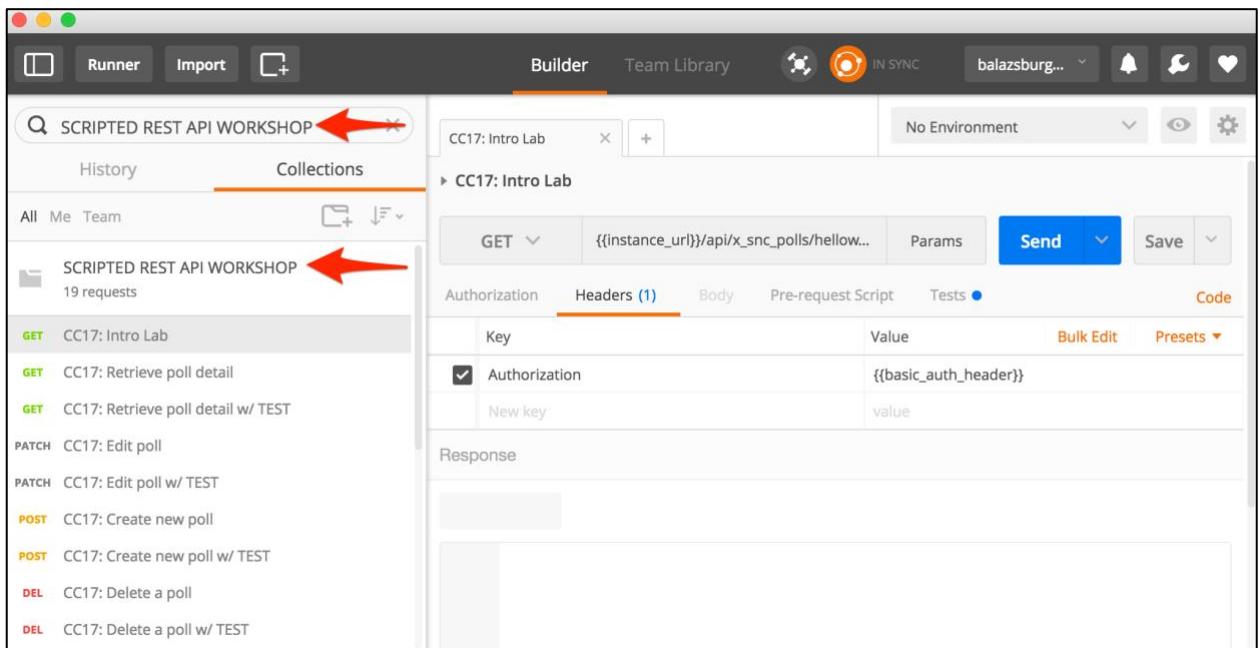
17. Open the **Postman** application on your laptop.
18. Import the Postman collection we will be using for this workshop from:
Postman Collection Link: <https://www.getpostman.com/collections/a689598a0d2920c2e570>
19. In Postman, click **Import**.



20. Paste the link to our Postman collection in the **Import from Link** and click **Import**.

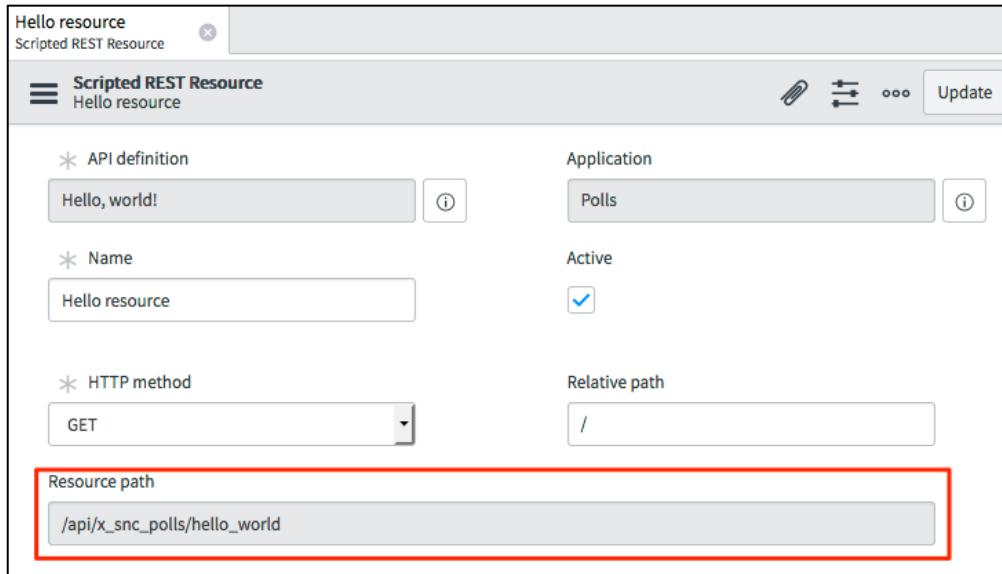


21. Verify you have the “Scripted REST API Workshop” collection loaded by searching for it in the navigator on the left hand side.



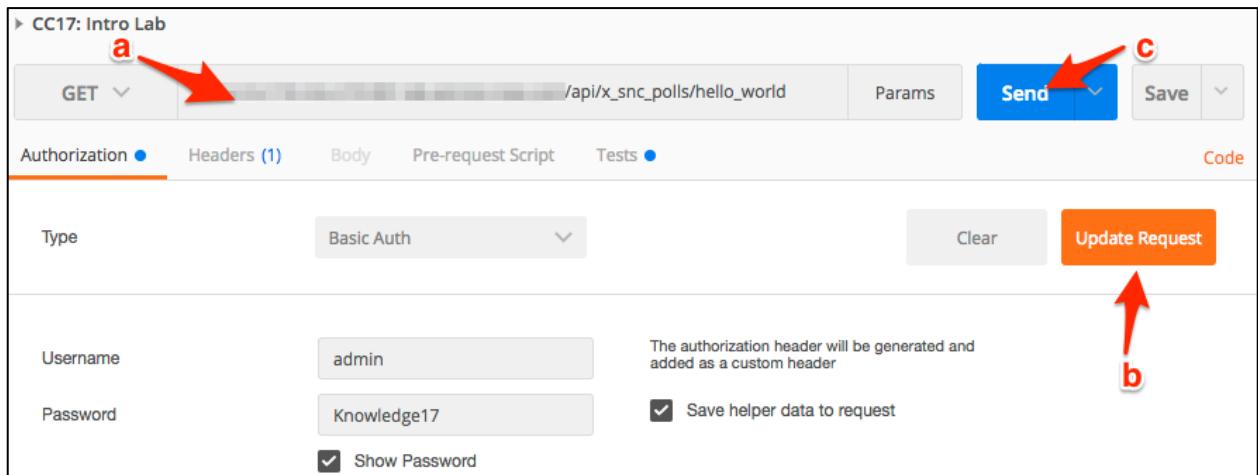
22. In the **Scripted REST API Workshop** select the **CC17: Intro Lab**.

- Replace {{instance_url}} with your lab instance URL (for example, https://my_instance.lab.service-now.com), and replace the resource URI with the resource from your Hello World Scripted REST API. Copy/paste the resource path from the **Resource path** field.



The screenshot shows the 'Scripted REST Resource' configuration page for a resource named 'Hello resource'. The 'Resource path' field contains '/api/x_snc_polls/hello_world' and is highlighted with a red border. Other fields include 'API definition' (Hello, world!), 'Application' (Polls), 'Name' (Hello resource), 'Active' (checked), 'HTTP method' (GET), and 'Relative path' (/).

- Click **Update Request**.
- Click **Send** to send the HTTP request.



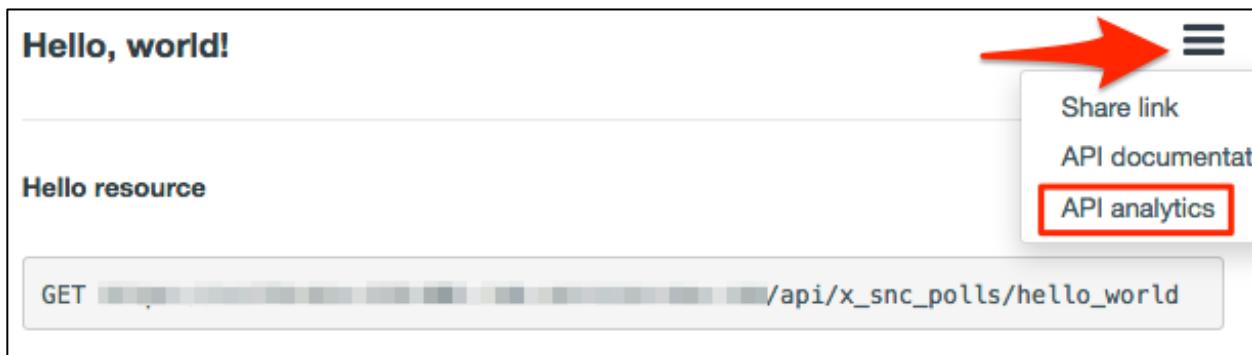
The screenshot shows a POSTMAN request configuration for the 'CC17: Intro Lab' collection. The request is a 'GET' to the URL '/api/x_snc_polls/hello_world'. The 'Authorization' tab is selected, showing 'Basic Auth' with 'Username' (admin) and 'Password' (Knowledge17). The 'Send' button is highlighted with a red arrow (labeled 'c'). A red arrow (labeled 'b') points to the 'Update Request' button. Other tabs include 'Headers (1)', 'Body', 'Pre-request Script', 'Tests', and 'Code'.

23. Validate response is successful by looking for the **200 OK** status code and message and that the response payload contains “Hello, world!”.

The screenshot shows the Postman application interface. At the top, there's a header bar with 'CC17: Intro Lab' and various buttons like 'Send' and 'Save'. Below the header, the request method is set to 'GET' and the URL is '/api/x_snc_polls/hello_world'. The 'Authorization' tab is selected, showing 'Basic Auth' with 'Username' as 'admin' and 'Password' as 'Knowledge17'. There's a note: 'The authorization header will be generated and added as a custom header'. A checked checkbox says 'Save helper data to request'. The 'Body' tab is selected, showing a JSON response with three lines of code: '1 {', '2 "result": "Hello, world!"', and '3 }'. To the right of the body panel, it says 'Status: 200 OK' and 'Time: 1332 ms'. Below the body panel are buttons for 'Pretty', 'Raw', 'Preview', 'JSON', 'Save Response', and a magnifying glass icon. Red arrows point from the bottom of the question text to the 'result' key in the JSON response and to the status bar at the bottom right of the Postman interface.

View API Analytics for Hello World

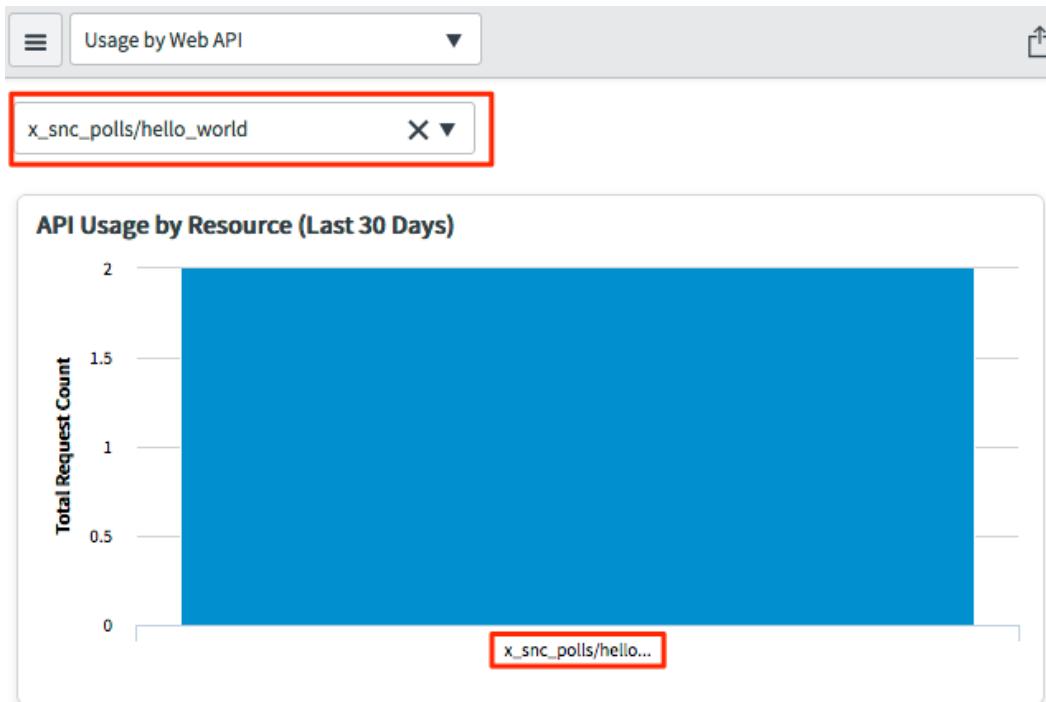
24. From **Explorer context menu**, or from Scripted REST API definition click **API Analytics**.



or

The screenshot shows the ServiceNow API Explorer interface. In the top left, there's a "Related Links" section with "Enable versioning", "Explore REST API", and "API analytics" (which is highlighted with a red box and a red arrow). Below this is a "Resources" list. The list header includes filters for "Name" and "Search", and columns for "Resources", "New", "Go to", "Name", "HTTP method", "Relative path", and "Resource path". One resource entry is shown: "Hello resource" (Name), "GET" (HTTP method), "/" (Relative path), and "/api/x_snc_polls/hello_world" (Resource path).

25. The **API Analytics** usage dashboard opens in a new browser tab, with the **Hello world API** pre-selected. Observe the API counts.



Note: There is up to a 60s delay between an API call and when it is reflected in API Analytics.

26. Close the REST API Explorer and API Analytics dashboard windows.

Lab 1 is complete. You are now ready to start lab 2.

Lab Goal

Having familiarized yourself with Scripted REST APIs in Lab 1, in Lab 2 we'll start building the "Polls" REST API that we'll use for the rest of this workshop. The Polls API you'll build provides a programmatic interface to interact with the Polls application on your ServiceNow instance.

The Polls app is a simple app that allows for the creation of Polls that allow participants to vote on answers to questions. Polls can have one or more questions associated with them. Questions can have one or more choices associated with them. As an example a simple poll could contain the question "What is your favorite color?". Choices that participants could choose would be; blue, red, yellow.

Lab 2 Building the Polls REST API

Create Lab 2 starting branch

1. In Studio, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab2-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab2-branch**
Create from Tag: **Lab2-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab2-branch**.
5. You are now ready to start **Lab 2**.

Create the Polls Scripted REST API

6. In Studio, click **Create New Application File**.
7. In the **Create New Application File** window, type **REST** in the filter then select **Scripted REST API** and click **Create**.
8. Give the **Scripted REST API** a name, then click **Submit**.

Name: **Poll**

API ID: **poll**

9. Click the related link **Enable versioning** to enable versioned URIs for the new API.

BEST PRACTICES

Do: Use versioning to control API changes.

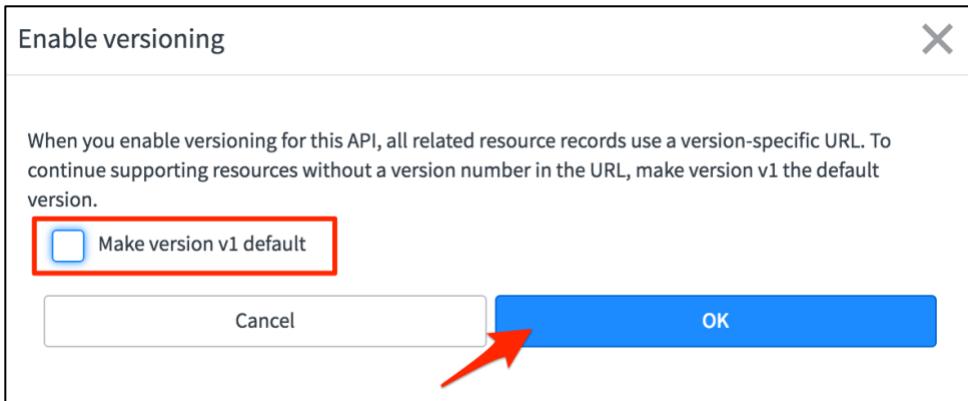
Do: Encourage clients to integrate against specific versions.

Don't: Make breaking changes in an existing version.

Do: Release a new API version when introducing new behaviors.

The screenshot shows the 'Scripted REST Service' configuration page for the 'Polls' application. The 'Name' field is set to 'Polls' and the 'API ID' field is set to 'poll'. The 'Active' checkbox is checked. Under 'Protection policy', there is a dropdown menu set to '-- None --'. Below this, there are tabs for 'Security', 'Content Negotiation', and 'Documentation', with 'Security' currently selected. Under 'Default ACLs', there is a lock icon. At the bottom of the form are 'Update' and 'Delete' buttons. A section titled 'Related Links' contains three items: 'Enable versioning', 'Explore REST API', and 'API analytics'. The 'Enable versioning' link is highlighted with a red arrow pointing to it.

10. In the Enable versioning popup, uncheck the **Make version v1 default** checkbox, then click **OK**.



11. A new tab **Versioning** appears. Click to review the versioning tab contents.

Service Versions				
	Version ID	Is default	Active	Deprecated
	v1	false	true	false

Note: The API versions are maintained here. Deactivate versions, mark a version **Is default=true** to allow non-versioned URIs to route to that version, or don't define a default version to force clients to specify the version when making requests to the API.

12. Add a **Resource** to the API. Click **New** on the **Resources** related list.

This resource will return the details of a specific poll.

13. Specify the following properties for the new resource and complete the script.

Name: **Retrieve poll detail**

API Version: **v1**

HTTP method: **GET**

Relative path: **{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab2_retrieve_poll_details

The screenshot shows the configuration of a Scripted REST Resource named 'Retrieve poll details'. The 'Name' field is highlighted with a red box. The 'API version' field is also highlighted with a red box and contains 'v1'. The 'HTTP method' dropdown is set to 'GET' and is highlighted with a red box. The 'Relative path' input field is highlighted with a red box and contains '{poll_id}'. The 'Resource path' input field below it contains '/api/x_snc_polls/v1/poll/{poll_id}'. The 'Script' section at the bottom shows the following code:

```
1 * (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var id = request.pathParams.poll_id;  
3     // Your logic here to retrieve poll details based on the ID  
4 } );
```

Click **Submit**.

Test with REST API Explorer

14. Click **Explore REST API**,

The screenshot shows the ServiceNow REST API Explorer interface. In the top left, under 'Related Links', the 'Explore REST API' option is highlighted with a red box. Below it, the 'Resources (1)' tab is selected. The main area displays a table with one row for the 'Retrieve poll detail' resource. The table columns include 'Name' (Retrieves poll detail), 'HTTP method' (GET), 'Relative path' (/api/x_snc_polls/v1/polls/{poll_id}), 'Resource path' (/api/x_snc_polls/v1/polls/{poll_id}), 'API version' (v1), and 'Active' (true). Navigation buttons like back, forward, and search are at the top right.

15. The “**Polls**” Scripted REST API is pre-selected in the Explorer menus and the **Retrieve Poll detail** resource is preselected.

16. Fill in sys id for a demo poll record and make a request.

To get the sys_id of demo record. Open the Polls module from navigator. Right click on existing record to copy sys_id.

The screenshot shows the ServiceNow Polls module. A record with the number POL0001002 is selected. A context menu is open over this record, with the 'Copy sys_id' option highlighted and a red arrow pointing to it. Other options in the menu include 'Filter Out', 'Copy URL to Clipboard', 'Show Matching', and 'Assign Tag'. The top navigation bar shows 'ServiceNow Service Management' and the user 'System Administrator'. The bottom right corner has page navigation buttons.

17. Fill in the sys_id on the REST API Explorer.

The screenshot shows the REST API Explorer interface. In the top left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1), all of which are highlighted with red boxes. Below these, a button labeled "Retrieve poll details (GET)" is also highlighted with a red box. To the right, under the heading "Poll", there is a description: "Retrieve poll details - Retrieve poll, questions, choices and votes polled by sysid". Below this is a GET request URL: "GET http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}". An arrow points from the "poll_id" placeholder in the URL to the "poll_id" parameter in the "Path parameters" section below. This section contains a table with two columns: "Name" and "Value". A row for "poll_id" has its "Value" cell highlighted with a yellow box and is also highlighted with a red box. The "Name" column for this row contains a red asterisk (*) indicating it is required.

Click **Send**.

18. Verify the response status code is **200-OK**.

The screenshot shows the REST API Explorer response details. Under the "Response" tab, the "Status code" is listed as "200 OK" in green. Below this, the "Headers" section lists various HTTP headers with their values. In the "Response Body" section, the JSON response is displayed:

```
{  
  "result": {  
    "name": "First poll",  
    "questions": [  
      {  
        "id": "2a46dae6134b1200ed373d62f244b041",  
        "question": "Favorite number",  
        "choices": [  
          {  
            "id": "3aa61ee6134b1200ed373d62f244b0dc",  
            "choice": "3",  
            "score": null  
          }  
        ]  
      }  
    ]  
  }  
}
```

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

19. Similar to creating the Lab2 starting branch, the completed lab can also be checked out from a tag (**Lab2-complete**) in Source control.
20. In **Studio**, navigate to **Source Control > Create Branch**.
21. In the pop-up window, enter a branch name, then select **Lab2-complete** from the **Create from Tag menu**, and click **Create Branch**.
Branch: **my-Lab2-branch-complete**
Create from Tag: **Lab2-complete**
22. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
23. Verify Studio is on branch **my-Lab2-branch-complete**.
24. You are now ready to continue with the next section of **Lab 2**.

Test with Postman

So far in this lab you’ve used Postman to make requests to ServiceNow REST APIs, Postman also allows you to write and execute tests that evaluate response from a REST API and provide you with Pass/Fail information based on your test and the response from the request. Let’s issue a request against the new resource **Retrieve poll detail** and write a few tests to verify the response we receive.

25. In Postman select the **CC17: Retrieve poll detail** request in the Scripted REST API Workshop collection. This request has been pre-built for you however you will need to update the {{instance_url}} and path including {{poll_id}} parameters in the URL replacing them with values from your lab instance. You will also need to update the Authorization section specifying your username and password.
 - instance_url: **URL of your lab instance**
 - poll_id: **Sys_id of a poll record in your lab instance**
 - Username: admin
 - Password: admin password for your lab instance

CC17: Retrieve poll detail

a {{instance_url}}/api/x_snc_polls/v1/poll/{{poll_id}}

b

Authorization Headers (1) Body Pre-request Script Tests ● Code

Type Basic Auth

Username admin The authorization header will be generated and added as a custom header

Password Knowledge17 Save helper data to request

Show Password

a. Replace {{instance_url}} with the URL of your lab instance (e.g., <https://mylabinstance.service-now.com>).
b. Replace {{poll_id}} with the sys_id of an existing Poll record in your ServiceNow instance.

26. After populating your credentials and replacing the parameters click **Update Request** and then **Send** the request.

27. Check that you've received a successful response. You should see a status of **200 OK** and a JSON payload that includes at least one poll as shown below.

CC17: Retrieve poll detail

GET /api/x_snc_polls/v1/poll/3c265ae6134b... Params Send Save

Authorization Headers (1) Body Pre-request Script Tests ● Code

Type Basic Auth

Username admin The authorization header will be generated and added as a custom header

Password Save helper data to request 

Show Password

Body Cookies Headers (10) Tests (3/3) Status: 200 OK Time: 240 ms

Pretty Raw Preview JSON  Save Response

```

1 {
2   "result": {
3     "name": "First poll",
4     "questions": [
5       {
6         "id": "2a46dae6134b1200ed373d62f244b041",
7         "question": "Favorite number",
8         "choices": [
9           {
10             "id": "3aa61ee6134b1200ed373d62f244b0dc",
11             "choice": "3",
12             "score": null
13           },
14           {
15             "id": "41a616e6134b1200ed373d62f244b0fc",
16             "choice": "4",
17             "score": null
18           }
19         ]
20       }
21     ]
22   }
23 }
```

28. This request should return a status code of **200 OK**, with a JSON payload that represents the poll we requested. In addition the content-type header in the response should be **application/json;charset=UTF-8** and our JSON payload should contain a **result** object. Let's see how we can use Postman to verify this for us with tests that will be run as part of the request.
29. In Postman, in the CC:17 Retrieve poll detail request, open the **Tests** tab by clicking on **Tests**. Here you can specify tests that will be run as part of each request.

```

1 tests["Status code is 200"] = responseCode.code === 200;
2 tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader("Content-Type") === "application/json;charset=UTF-8";
3 tests["Response Body Contains result"] = responseBody.has("result");

```

30. Postman has its own simple syntax for declaring tests. You can find out more about this syntax at the Postman website. For this lab we've provided you with 3 tests that validate that:
- the response status code is **200**
 - the response includes a content-type header with a value of **application/json;charset=UTF-8**
 - the response body contains the text '**result**'
31. Now update your request in Postman to run these tests. Copy the test script from the following URL and paste it into the **Tests** area in Postman.
- Postman test script: http://bit.ly/CC17_ScriptedRESTAPI_Lab2_postman_test_script

32. After copying click **Send** to issue the request. Now that we have tests specified as part of our request test results will be displayed in the response area. If all tests passed you will see a '(3/3)' in the header and then a green **PASS** image next to each test as shown below.

The screenshot shows the Postman interface with the 'Tests' tab selected, indicated by an orange arrow. The status bar at the top right shows 'Status: 200 OK' and 'Time: 118 ms'. Below the tabs, there are three test results, each with a green 'PASS' icon and a description: 'Status code is 200', 'Content-Type is application/json; charset=UTF-8', and 'Response Body Contains result'. A red arrow points upwards from the bottom of the page towards the 'Tests' tab in the screenshot.

Save your request in Postman. You now have a saved request in Postman that allows you to easily issue a request to your 'Retrieve poll detail' resource and which will run test the response to validate that it includes the correct status code, header, and payload content. These were simple test cases but Postman will allow you to define more advanced test cases to verify you are receiving the correct response from your REST API.

BEST PRACTICES

Do: Define test cases for each of your APIs resources to validate that the response is formatted correctly and that the response contains the intended content. Building test cases as part of your development process will help insure you're building the API as you designed it and provide you with a set of tests that can be run over time as you make changes to guarantee that your interface has not changed unintentionally.

33. Close the REST API Explorer and API Analytics dashboard windows.

Lab 2 is complete. You are now ready to begin lab 3.

Lab Goal

In Lab 3 you'll continue building out the REST API for the **Polls** application adding resources to support creating a new poll, editing an existing poll, and voting in a poll. In building out this additional functionality you will further use and familiarize yourself with the Request and Response APIs that allow you to interact with the request that your REST API receives and build the response that your REST API will return.

Lab 3 Request & Response API

Create the Lab 3 starting branch

1. In Studio, navigate to Source Control > Create Branch.
2. In the pop-up window, enter a branch name, then select Lab3-start from the Create from Tag menu, and click **Create Branch**.

Branch: **my-Lab3-branch**
Create from Tag: **Lab3-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab3-branch**.
5. You are now ready to start **Lab 3**.

Create New Resource in Polls API - Create a poll

6. The ‘Create a poll’ resource will be used to create a new poll in the ‘Polls’ application. Open the Polls API in studio and click **New** on the **Resources** related list to create a new resource.

The screenshot shows the ServiceNow API Studio interface. On the left, the Application Explorer sidebar lists various categories like Data Model, Forms & UI, Server Development, Access Control, Navigation, and Integrations. Under Integrations, Scripted REST APIs is expanded, showing a 'Hello, world!' example and a 'Poll' resource. A red arrow points from the 'Poll' link to the main content area. The main content area displays a table for the 'Poll' resource, which currently has one row with values: Name (v1), Active (true), and Poll (false). Below the table are 'Update' and 'Delete' buttons. A 'Related Links' section includes 'Add new version', 'Explore REST API', and 'API analytics'. At the bottom, a table lists resources under the 'API definition = Poll'. The 'New' button in the top navigation bar of this table is highlighted with a blue box. The table columns are: Name, HTTP method, Relative path, Resource path, API version, and Active. One row is shown: Name is 'Retrieve poll details', HTTP method is 'GET', Relative path is '/[poll_id]', Resource path is '/api/x_snc_polls/v1/poll/[poll_id]', API version is 'v1', and Active is 'true'.

7. Specify the following properties for the new resource.

Name: **Create new poll**

API Version: **v1**

HTTP method: **POST**

Relative path: **/**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll

The screenshot shows the 'Create new poll' configuration page in the ServiceNow interface. The top navigation bar indicates it's a 'Scripted REST Resource'. The main form has the following fields:

- API definition:** Poll (Name field is highlighted with a red box)
- Application:** Polls (Active checkbox is checked)
- API version:** v1 (highlighted with a red box)
- Request routing:** HTTP method is POST, Relative path is / (both highlighted with red boxes)
- Implement the resource:** A script editor containing the following code (highlighted with a red box):

```
1 * (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var data = request.body.data;  
3     var pollHelper = new x_snc_polls.PollData_Creator();  
4     var groupId = getUserId(data.usergroup);  
5  
6     // Create Poll record  
7     var pollRecord = pollHelper.createPoll(data, groupId);  
8  
9     // End of function definition  
10}
```

Click **Submit**.

Test with REST API Explorer

8. Open 'Create new poll' resource and Click **Explore REST API**.

The screenshot shows the REST API Explorer interface for the 'Create new poll' resource. The top navigation bar includes 'Create new poll' (highlighted with a red box), 'Scripted REST Resource', and 'Scripted REST Resource' (with a red box). The main content area displays security settings: 'Requires authentication' (checked) and 'Requires ACL authorization' (unchecked). Below these settings are 'Update' and 'Delete' buttons. A 'Related Links' section contains 'Explore REST API' (highlighted with a red box) and 'API analytics'. At the bottom, a 'Request Header Associations' panel shows a table with one row: 'API resource = Create new poll', 'API request header', 'Example value', and 'Is required'. A search bar at the top of this panel is set to 'Search for text'.

9. Create New poll resource is shown in the REST API Explorer. Fill in request body in the raw tab under Request body section.

The screenshot shows the REST API Explorer interface. In the top left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1). Below these is a red box around the 'Create new poll (POST)' button. To the right, the 'Poll' section is displayed with the sub-section 'Create new poll'. A URL 'POST http://10.11.91.87:16001/api/x_snc_polls/v1/poll' is shown. Under 'Prepare request', there are sections for 'Query parameters' (with 'Add query parameter' button) and 'Request headers' (with 'Name', 'Value', and 'Description' columns for Request format, Response format, and Authorization). The Authorization dropdown has 'Send as me' selected.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll_sample_request

10. Copy the sample payload into the 'Raw' tab.

The screenshot shows the 'Request Body' tab in the REST API Explorer. At the top, there are two tabs: 'Builder' (selected) and 'Raw'. A red arrow points from the text above to the 'Raw' tab. Below the tabs is a code editor containing a JSON payload. This payload defines a poll with a user group ('Software'), a question ('what is your favorite car'), and four choices: 'Audi', 'BMW', 'Corvette', and 'Tesla'. The JSON is highlighted with a red box. At the bottom of the tab, there are 'Send' and 'Clear response' buttons, and a footer with links to various scripting languages.

Click **Send**.

11. Verify response status code is **201 Created**

Response	
Status code	201 Created
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 15:55:46 GMT
Expires	0
Location	http://10.11.91.87:16001/api/x_snc_polls/v1/poll/83e342dab1321200964f2e4c16efa08f
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "result": { "number": "POL0001001", "name": "Second poll" } }	

Create tests in Postman

12. In Postman select the ‘CC17: Create new poll’ request’. This is a pre-built request that already contains an appropriately formatted payload for the ‘Create new poll’ resource.

The screenshot shows a Postman collection named 'SCRIPTED REST API WORKSHOP' containing 19 requests. The requests listed are:

- GET CC17: Intro Lab
- GET CC17: Retrieve poll detail
- GET CC17: Retrieve poll detail w/ TEST
- PATCH CC17: Edit poll
- PATCH CC17: Edit poll w/ TEST
- POST CC17: Create new poll** (highlighted)
- POST CC17: Create new poll w/ TEST
- DEL CC17: Delete a poll
- DEL CC17: Delete a poll w/ TEST
- POST CC17: Vote in poll

13. Update the request replacing the {{instance_url}} and authorization credentials appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request.

14. As you saw when you tested with the REST API Explorer a successful response will include a **201** status code, a JSON payload that includes the number for the newly created poll, and the response headers include a ‘Location’ header that provides the URL for this newly created record. Let’s add tests in Postman that verify that the following details for in the response:

- Response status code is 201
- Response headers include Location
- Response headers include Content-Type of application/json;charset=UTF-8
- Response body contains the text ‘number’
- Response body contains the text ‘name’

Update the request in Postman to include the following:

```
tests["Status code is 201"] = responseCode.code === 201;
tests["Location Header is present"] = postman.getResponseHeader("Location");
tests["Response Body Contains number"] = responseBody.has("number");
tests["Response Body Contains name"] = responseBody.has("name");
tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader("Content-Type") === "application/json;charset=UTF-8";
```

For ease you can also copy these from:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll_test_script

15. After adding the tests above save your request in Postman and Send the request. In the response you should see the following tests and results.

The screenshot shows the Postman interface for a POST request to `https://[REDACTED].service-now.com/api/x_snc_global_polls/v1/poll`. The 'Tests' tab is selected, displaying the following test script:

```
1 tests["Status code is 201"] = responseCode.code === 201;
2 tests["Location Header is present"] = postman.getResponseHeader("Location");
3 tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader
    ("Content-Type") === "application/json;charset=UTF-8";
4 tests["Response Body Contains number"] = responseBody.has("number");
5 tests["Response Body Contains name"] = responseBody.has("name");
```

The results section shows 5/5 tests passed:

- PASS Status code is 201
- PASS Location Header is present
- PASS Content-Type is application/json;charset=UTF-8
- PASS Response Body Contains number
- PASS Response Body Contains name

On the right side, a sidebar titled 'SNIPPETS' lists various assertions and helpers:

- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check
- Response time is less than 200ms
- Set a global variable

Create New Resource in Polls API – Edit poll

The ‘Edit poll’ resource will be used to modify an existing poll record in the ‘Polls’ application.

16. Open the Polls API in studio and click New on the Resources related list to create a new resource.

17. Specify the following properties for the new resource.

Name: **Edit poll**

API Version: **v1**

HTTP method: **PATCH**

Relative path: **{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll

The screenshot shows the 'Scripted REST Resource' configuration screen. The 'Name' field is set to 'Edit poll' and the 'API version' field is set to 'v1'. The 'HTTP method' is 'PATCH' and the 'Relative path' is '{poll_id}'. The 'Resource path' is '/api/x_snc_polls/v1/poll/{poll_id}'.

Field	Value
Name	Edit poll
API definition	Poll
Application	Polls
API version	v1
HTTP method	PATCH
Relative path	{poll_id}
Resource path	/api/x_snc_polls/v1/poll/{poll_id}

Click **Submit**.

Test with REST API Explorer

Open 'Edit poll' resource and Click **Explore REST API** in related actions.

18. **Edit poll** resource is preselected in API Explorer.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace ('x_snc_polls'), API Name ('Poll'), and API Version ('v1'). Below these are three buttons: 'Create new poll (POST)', 'Retrieve poll details (GET)', and 'Edit poll (PATCH)'. The 'Edit poll (PATCH)' button is highlighted with a red box. On the right, under the 'Poll' section, a 'PATCH' request is shown with the URL 'http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}'. Below this, the 'Prepare request' section shows a path parameter 'poll_id' with the value 'ddee64b9443a1200964fac543127a1ab' highlighted with a yellow box and a red arrow pointing to it from the left side of the screen.

a. Fill in request body in raw tab under Request body section.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll_sample_request

b. Copy the sample payload into the 'Raw' tab.

The screenshot shows the 'Request Body' tab in the REST API Explorer. It has two tabs: 'Builder' (which is active) and 'Raw'. The 'Builder' tab contains a JSON payload:

```
[{"question": "what is your favorite film", "choices": [{"choice": "Avatar"}, {"choice": "Avengers"}, {"choice": "Toy Story"}, {"choice": "Frozen"}]}]
```

 Below the tabs is a 'Send' button. At the bottom of the page, there are links for [ServiceNow Script], [cURL], [Python], [Ruby], [JavaScript], [Perl], and [Powershell].

Click **Send**.

19. Verify response status code is **204-No content**.

Response

Status code	204 No Content
Headers	
Content-Encoding	gzip
Date	Sun, 17 Apr 2016 16:17:06 GMT
Server	ServiceNow
X-Is-Logged-In	true
Response Body	
...	

Create tests in Postman

20. In Postman select the ‘CC17:Edit poll’ request. This is a pre-built request that already contains an appropriately formatted payload for calling the ‘Edit poll’ resource.

The screenshot shows the Postman interface with a sidebar on the left. The sidebar lists a folder named "SCRIPTED REST API WORKSHOP" which contains 19 requests. Below the folder, there is a list of individual requests, each with a method (e.g., GET, PATCH, POST, DELETE) and a name (e.g., "CC17: Intro Lab", "CC17: Edit poll"). The "PATCH CC17: Edit poll" request is highlighted with a gray background, indicating it is selected.

- SCRIPTED REST API WORKSHOP
19 requests
- GET CC17: Intro Lab
- GET CC17: Retrieve poll detail
- GET CC17: Retrieve poll detail w/ TEST
- PATCH CC17: Edit poll**
- PATCH CC17: Edit poll w/ TEST
- POST CC17: Create new poll
- POST CC17: Create new poll w/ TEST
- DELETE CC17: Delete a poll
- DELETE CC17: Delete a poll w/ TEST
- POST CC17: Vote in poll
- POST CC17: Vote in poll w/ TEST
- GET CC17: Retrieve poll results

21. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you've updated those values save and then send the request.

As you saw when you tested with the REST API Explorer a successful response will include a **204** status code and an empty payload. Let's add tests in Postman that verify that the following details in the response:

- Response status code is 204
- Response payload is empty

22. Update the request in Postman to include the following:

```
tests["Status code is 204"] = responseCode.code === 204;  
tests["Body is empty"] = responseBody === "";
```

For ease you can also copy these from:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll_test_script

23. After adding the tests above save your request in Postman and Send the request. In the response you should see the following tests and results.

The screenshot shows the Postman interface for a PATCH request to `https://.../v1/poll/dad383c41`. The 'Tests' tab is active, displaying the following test scripts:

```
1 tests["Status code is 204"] = responseCode.code === 204;  
2 tests["Body is empty"] = responseBody === "";
```

Below the tests, the 'Tests (2/2)' tab is selected, showing the results of the executed tests:

- PASS Status code is 204
- PASS Body is empty

A sidebar on the right contains a list of snippets:

- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check
- Response time is less than 200ms
- Set a global variable

Create New Resource in Polls API – Vote in poll

24. The ‘Vote in poll’ resource will be used to cast a vote for an answer to a specific question or set of questions that are part of a poll in the ‘Polls’ application. Open Polls API in studio and add a **Resource** to the API. Click **New** on the **Resources** related list.

The screenshot shows the ServiceNow API Studio interface for creating a new resource. The top section displays the API definition and application details:

- Name:** Vote in poll
- Application:** Polls
- API version:** v1
- Active:** checked

The **Request routing** section contains the following configuration:

- HTTP method:** POST
- Relative path:** /{poll_id}/vote
- Resource path:** /api/x_snc_polls/v1/poll/{poll_id}/vote

The **Implement the resource** section includes a script editor with the following code:

```
1 * (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var pollId = request.pathParams.poll_id;  
3     var pollHelper = new x_snc_polls.PollData_Creator();  
4  
5     // Validate if poll record exists  
6     var pollRecord = new GlideRecord("x_snc_polls_poll");  
7 }
```

Specify the following properties for the new resource.

Name: **Vote in poll**

API Version: **v1**

HTTP method: **POST**

Relative path: **{poll_id}/vote**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_vote_in_poll

The screenshot shows the 'API definition' screen for a 'Poll' API. Key fields include:

- Name:** Vote in poll
- Application:** Polls
- API version:** v1
- Active:** checked

Request routing:

- HTTP method:** POST
- Relative path:** /{poll_id}/vote
- Resource path:** /api/x_snc_polls/v1/poll/{poll_id}/vote

Implement the resource:

Script content:

```

1 v
2
3     var pollId = request.pathParams.poll_id;
4     var pollHelper = new x_snc_polls.PollData_Creator();
5
6     // Validate if poll record exists
7     var pollRecord = new GlideRecord("x_snc_polls_poll");
8
9
10    var voteData = request.body.data.votes;
11
12    // Record votes
13    pollHelper.voteInPoll(voteData, pollId);
14
15    // Set response details
16    response.setStatus(201);
17    response.setContentType("application/json");
18    var responseBody = '{"message": "Voting successful"}';
19    response.getWriter().writeString(responseBody);
20
21 })(request, response);

```

NOTE: Observe the custom response string being written to the response using the 'getStreamWriter' method. The getStreamWriter method is used to produce a custom response in Scripted REST APIs and allows you (the API creator) to precisely specify the format of the response. It is important to set content type and status code if writing to stream directly.

The screenshot shows the 'Script' editor with the following content:

```

1 v
2
3     var pollId = request.pathParams.poll_id;
4     var pollHelper = new x_snc_polls.PollData_Creator();
5
6     // Validate if poll record exists
7     var pollRecord = new GlideRecord("x_snc_polls_poll");
8     pollRecord.get(pollId);
9
10    var voteData = request.body.data.votes;
11
12    // Record votes
13    pollHelper.voteInPoll(voteData, pollId);
14
15    // Set response details
16    response.setStatus(201);
17    response.setContentType("application/json");
18    var responseBody = '{"message": "Voting successful"}';
19    response.getWriter().writeString(responseBody);
20
21 })(request, response);

```

A red box highlights the final part of the script where the response is set using the getStreamWriter method.

Protection policy: --None--

Click **Submit**.

Test with REST API Explorer

25. Open 'Vote in poll' resource and Click **Explore REST API** in related actions.

26. **Vote in poll** resource is preselected in API Explorer. Fill in request body in raw tab under Request body section.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1). Below these are four buttons: 'Create new poll (POST)', 'Retrieve poll details (GET)', 'Edit poll (PATCH)', and 'Vote in poll (POST)'. The 'Vote in poll (POST)' button is highlighted with a red box. On the right, the 'Poll' section is displayed with the sub-section 'Vote in poll - Answer poll'. A POST request URL is shown: `POST http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}/vote`. Under 'Prepare request', there are sections for 'Path parameters' and 'Query parameters'. In the 'Path parameters' section, there is a table with one row: Name 'poll_id' and Value 'ddee64b9443a1200964fac543127a1ab'. This row is also highlighted with a red box. A red arrow points from the 'Value' column of this table to the 'Value' column in the 'Path parameters' table.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_vote_in_poll_sample_request

NOTE: you will need to update the 'poll_id' to be that of a specific poll that exists in the Polls application on your lab instance.

The screenshot shows the 'Request Body' tab in the REST API Explorer. It has two tabs: 'Builder' and 'Raw'. The 'Raw' tab is selected and contains a red box around the following JSON payload:

```
{ "votes": [{"question_id": "d5f383c4137612006ae13d62f244b056", "vote": "Yellow"}]}
```

Below the tabs, there are five buttons: [ServiceNow Script], [cURL], [Python], [Ruby], and [JavaScript]. At the bottom left is a 'Send' button.

Click **Send**.

27. Verify response status code is **201 Created**.

Response

Status code	201 Created
Headers	
Content-Encoding	gzip
Content-Type	application/json
Date	Sun, 17 Apr 2016 17:27:27 GMT
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true

Response Body

```
{  
  "message": "Voting successful"  
}
```

Create tests in Postman

27. In Postman select the ‘CC17:Vote in poll’ request. This is a pre-built request that already contains an appropriately formatted payload for calling the ‘Vote in poll’ resource.
28. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request.
29. As you saw when you tested with the REST API Explorer a successful response will include a **201** status code and an JSON payload informing you that voting was successful. Add tests in Postman that verify that the following details in the response:
 - Response status code is 201
 - Response headers include Content-Type of application/json; charset=UTF-8
 - Response body contains the text: “Voting successful”

30. You are on your own to create these tests in Postman. You can refer back to the tests you've created in the previous steps for help.

31. Once you've added the tests save the request and send it. If you were successful you should see all the tests passing.

The screenshot shows the 'Tests' tab in Postman with three test cases listed:

- PASS Status code is 201
- PASS Content-Type is application/json;charset=UTF-8
- PASS Response Body Contains 'Voting successful'

Note: If you are really stuck here you can refer to the pre-built request in the Postman collection named "CC17: Vote in poll w/ TEST" to see this request with tests fully specified.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

32. Similar to creating the Lab3 starting branch, the completed lab can also be checked out from a tag (**Lab3-complete**) in Source control.

33. In **Studio**, navigate to **Source Control > Create Branch**.

34. In the pop-up window, enter a branch name, then select **Lab3-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab3-branch-complete**

Create from Tag: **Lab3-complete**

35. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

36. Verify Studio is on branch **my-Lab3-branch-complete**.

Lab 3 is complete. You are now ready to begin lab 4.

Lab Goal

In Lab 4 you'll continue building out the REST API for the **Polls** application adding resources to support retrieving the results of a poll which includes details of individual votes as well as the ability to delete a poll. These operations expose functionality that should be restricted to users with an additional role so that we can limit access to see how individual users voted as well as be able to delete polls.

Scripted REST APIs allow you to specify ACLs that requestors must have to be able to make a request both at the API and Resource level. These ACLs can then be associated users or groups via the standard access control mechanism in ServiceNow.

Scripted REST APIs allow you to configure, at both the API and Resource level, if a requestor needs to **authenticate** (via Basic Auth or OAuth2.0) to ServiceNow to make requests. In addition, you can configure if the requestor must be authorized, via specific ACLs, to make a request to your API.

In building out these additional resources you will familiarize yourself with how you can use the security features of Scripted REST APIs to secure your REST API.

Lab 4 Enforcing Security

Create Lab 4 starting branch

1. In **Studio**, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab4-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab4-branch**
Create from Tag: **Lab4-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab4-branch**.
5. You are now ready to start **Lab 4**.

Create New Resource in Polls API – Retrieve poll results

6. Open Polls API from studio. Add a **Resource** to the API. Click **New** on the **Resources** related list.

Give the resource a **Name**. Complete the script.

Name: **Retrieve poll results**

API Version: **v1**

HTTP method: **GET**

Relative path: **/{poll_id}/results**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab4_retrieve_poll_results

NOTE: Notice that the script is using the GlideRecordSecure API.

The screenshot shows the 'Scripted REST Resource' configuration page. The 'Name' field is set to 'Retrieve poll results'. The 'API version' dropdown is set to 'v1'. The 'HTTP method' is 'GET' and the 'Relative path' is '/{poll_id}/results'. The 'Script' section contains the following code:

```
1 v (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var id = request.pathParams.poll_id;  
3     var pollHelper = new x_snc_polls.PollData_Retriever();  
4     var pollRecord = new GlideRecordSecure("x_snc_polls_poll");  
5     pollRecord.get(id);  
6 }
```

7. Enable ACL authorization on the resource by setting an ACL. ACL settings are available under Security tab

Requires ACL authorization: **checked**

ACLs: Click to unlock, and browse to select the **Poll Manager** ACL

The screenshot shows the 'Security' tab of the API Resource configuration. The 'Requires ACL authorization' checkbox is checked. The 'ACLs' dropdown is set to 'Poll Manager'. Both the checkbox and the dropdown are highlighted with a red box.

NOTE: Only ACLs of type REST Endpoint can be used.

Click **Submit**.

BEST PRACTICES

Do: Use the *GlideRecordSecure API* in your Scripted REST API Resource scripts to ensure that you are enforcing existing access controls on the requesting user when interacting with ServiceNow records.

Do: Test your access controls, both Authentication and Authorization, before making your API available to consumers.

Test with REST API Explorer

8. Open 'Retrieve poll results' resource and Click **Explore REST API** in related actions.
9. **Retrieve poll results** resource is preselected in API Explorer. Fill in sys_id of poll.

The screenshot shows the REST API Explorer interface. The 'Poll' resource is selected. The 'Retrieve poll results' action is highlighted with a red box. In the 'Prepare request' section, the 'poll_id' parameter is filled with the value 'ddee64b9443a1200964fac543127a1ab', which is also highlighted with a red box.

Click **Send**.

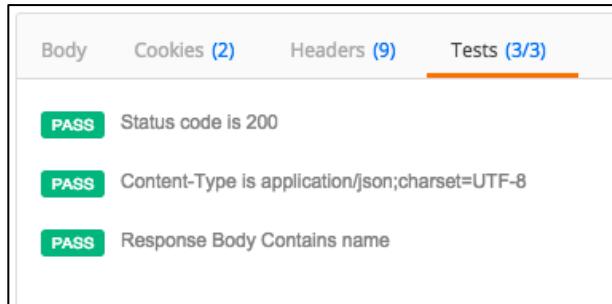
10. Verify response status code is **200-Ok**.

Response	
Status code	200 OK
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 17:51:53 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "result": { "name": "First Poll", "questions": [{ "question": "Favorite number", "users": [] }, { "question": "Favorite color", "users": [{ "user": "admin", "answer": "Yellow" }] }] } }	

Create tests in Postman

11. In Postman select the ‘CC17: Retrieve poll results’ request. This is a pre-built request that will make a request to the ‘Retrieve poll results’ resource.
12. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request. If the request is successful (200 OK) you will see a response similar to the one you saw when testing in the REST API Explorer.
13. Now that you’ve made a successful request add tests to your Postman requests to validate the request matches the expected results. Add tests that verify the following details in the response:
 - Response status code is 200
 - Response headers include Content-Type of application/json;charset=UTF-8
 - Response body contains the text: “name”
14. You are on your own to create these tests in Postman. You can refer back to the tests you’ve created in the previous exercises for help.

15. Once you've added the tests save the request and send it. If you were successful you should see all the tests passing.



A screenshot of the Postman interface showing test results. The tabs at the top are Body, Cookies (2), Headers (9), and Tests (3/3). The Tests tab is selected, indicated by an orange underline. Below the tabs, there are three test cases listed:

- PASS Status code is 200
- PASS Content-Type is application/json;charset=UTF-8
- PASS Response Body Contains name

Note: If you are really stuck here you can refer to the pre-built request in the Postman collection named “CC17: Retrieve poll results w/ TEST” to see this request with tests fully specified.

Create New Resource in Polls API – Delete poll

16. Open Polls API from studio. Add a **Resource** to the API. Click **New** on the **Resources** related list.

17. Give the resource a **Name**. Complete the script.

Name: **Delete poll**

API Version: **v1**

HTTP method: **DELETE**

Relative path: **/{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab4_delete_poll

The screenshot shows the 'Scripted REST Resource' configuration page. The 'Name' field is set to 'Delete poll'. The 'Application' dropdown is set to 'Polls' and the 'API version' is 'v1'. The 'HTTP method' is 'DELETE' and the 'Relative path' is '/{poll_id}'. The 'Script' editor contains the following code:

```
1 v  (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2 v     var pollId = request.pathParams.poll_id;  
3 v     var pollRecord = new GlideRecordSecure("x_snc_polls_poll");  
4 v     pollRecord.get(pollId);  
5 v     if (pollRecord.isValidRecord()) {  
6 v         pollRecord.deleteRecord();  
7 v     }  
8 v }
```

18. Enable ACL authorization on the resource by setting an ACL. ACL settings available under Security tab.

Requires ACL authorization: **checked**

ACLs: Click to unlock, and browse to select the **Poll Manager** ACL

Resources can specify security settings that override the parent settings. By default resources 'Require authentication' but do not 'Require ACL authorization'. To make a resource public, meaning no authentication is required to access the resource, uncheck 'Requires authentication'. To require authorization, select the 'Requires ACL authorization' check box and select an ACL record(s). Leave the 'ACL' field blank to enforce the 'Default ACLs' from the parent API. Access is granted if at least one matching ACL record is found.

More info

Requires authentication

Requires ACL authorization

ACLs Poll Manager

NOTE: Only ACLs of type REST Endpoint can be used

Click **Submit**.

Type REST_Endpoint

Operation execute

Admin overrides

Name Poll Manager

Description

Definition

Access Control Rules allow access to the specified resource if *all three* of these checks evaluate to true:
The user has one of the roles specified in the Role list, or the list is empty.
Conditions in the Condition field evaluate to true, or conditions are empty.
The script in the Script field (advanced) evaluates to true, or sets the variable "answer" to true, or is empty.

The three checks are evaluated independently in the order displayed above.

More Info

Requires role

Role x_snc_polls.poll_manager

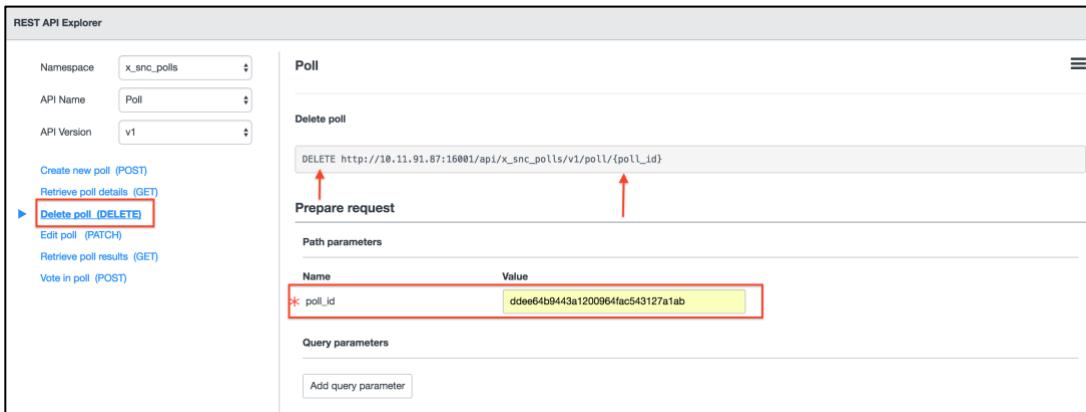
Condition Not a valid table name (empty)

Scripted REST Resources	Go to	Name	Search	1 to 2 of 2	Next	Previous	First	Last
ACLs = Poll Manager								
Name	HTTP method	Active	Relative path	Resource path	API definition	API version		
Delete poll	DELETE	true	/poll_id	/api/x_snc_polls/v1/poll/{poll_id}	Poll	v1		
Retrieve poll results	GET	true	/poll_id/results	/api/x_snc_polls/v1/poll/{poll_id}/results	Poll	v1		

NOTE: REST_Endpoint type ACLs (as shown above) are used to restrict access to Scripted REST API Resources. The 'Poll Manager' ACL has been specified on the 'Delete poll' resource and restricts access to this resource to users who have the role 'x_snc_polls.poll_manager'. **Only** users with this role can make requests to the 'Delete poll' resource.

Test with REST API Explorer

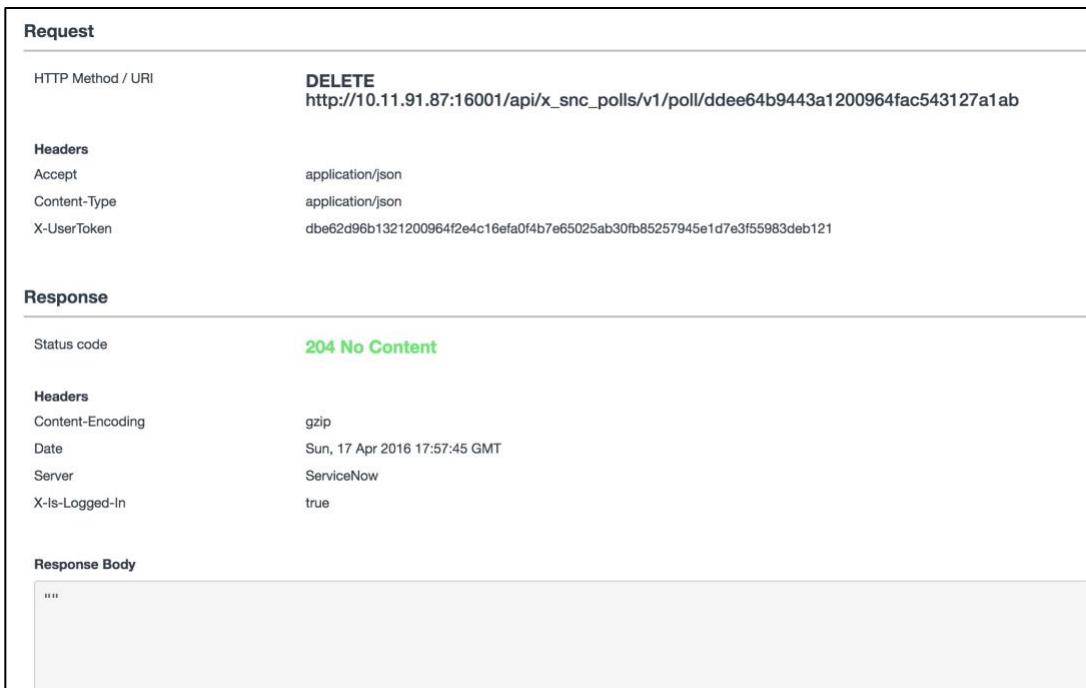
19. Open Delete poll resource and click **Explore REST API** in related actions.
20. **Delete poll** resource is preselected in API Explorer. Fill in request body in raw tab under Request body section.



The screenshot shows the REST API Explorer interface. On the left, there's a sidebar with various API actions: Create new poll (POST), Retrieve poll details (GET), Delete poll (DELETE) (which is highlighted with a red box), Edit poll (PATCH), Retrieve poll results (GET), and Vote in poll (POST). The main area is titled 'Poll' and shows the 'Delete poll' action. It includes a 'DELETE' button, a 'Prepare request' section, and a 'Path parameters' table. In the 'Path parameters' table, there's a row for 'poll_id' with a value 'ddee64b9443a1200964fac543127a1ab' highlighted with a yellow box. Below the table is a 'Query parameters' section with a 'Add query parameter' button.

Click **Send**.

21. Verify response status code is **204-No content**.



The screenshot shows the REST API Explorer interface with the 'Request' tab selected. It displays an HTTP DELETE request to 'http://10.11.91.87:16001/api/x_snc_polls/v1/poll/ddee64b9443a1200964fac543127a1ab'. The 'Headers' section includes Accept: application/json, Content-Type: application/json, and X-UserToken: dbe62d96b1321200964f2e4c16efa0f4b7e65025ab30fb85257945e1d7e3f55983deb121. The 'Response' tab is selected and shows the 'Status code' as '204 No Content'. The 'Headers' section of the response includes Content-Encoding: gzip, Date: Sun, 17 Apr 2016 17:57:45 GMT, Server: ServiceNow, and X-Is-Logged-In: true. The 'Response Body' section is empty.

Create tests in Postman

22. In Postman select the 'CC17:Delete poll' request. This is a pre-built request that will make a request to the 'Delete poll' resource.

23. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. And make sure that the admin user has the ‘x_snc_polls.poll_manager’ role. Once you’ve updated those values save and then send the request. If the request is successful (204 No Content) you will see a response similar to the one you saw when testing in the REST API Explorer.

24. Add tests that verify the following details in the response:

- Response status code is 204
- Response body is empty

25. After adding these tests issue the request and verify that your tests are passing as shown below.

Body	Cookies (2)	Headers (5)	Tests (2/2)
			PASS Status code is 204
			PASS Response Body is empty

26. Create a new poll in your instance and then update this request in Postman to use the new poll id and update the user credentials to use a user that **does not have** the ‘x_snc_polls.poll_manager’ role. Update the request in Postman and send the request.

27. **NOTE:** you have been deleting polls so you may need to go back and create some additional poll records in your instance so that there are polls that you can delete (hint use insert and stay to quickly create new polls for testing).

28. Send your updated request now and verify that for a user that when making a request with a user that **does not have** the ‘x_snc_polls.poll_manager’ role you receive a status code of **403 Forbidden** and that your test fail case ‘Status code is 204’ **fails** as shown below.

Body	Cookies (2)	Headers (10)	Tests (1/2)
			FAIL Status code is 204
			PASS Response Body is empty

29. **NOTE:** If you are really stuck here you can refer to the pre-built request in the Postman collection named “CC17: Delete a poll w/ TEST” to see this request with tests fully specified.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

30. Similar to creating the Lab4 starting branch, the completed lab can also be checked out from a tag (**Lab4-complete**) in Source control.
31. In **Studio**, navigate to **Source Control > Create Branch**.
32. In the pop-up window, enter a branch name, then select **Lab4-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab4-branch-complete**

Create from Tag: **Lab4-complete**

33. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
34. Verify Studio is on branch **my-Lab4-branch-complete**.
35. Close the REST API Explorer and API Analytics dashboard windows.

Lab 4 is complete. You are now ready to begin lab 5.

Lab Goal

Versioning a REST API is a common task when you want to introduce new functionality or behaviors to your REST API but don't want to break existing clients. Scripted REST APIs support easily versioning your resources. With versioning support you can quickly create new versions of existing resources to introduce new functionality. You have the ability to specify what version of a resource is the default and to which requests will be routed if the client does not specify a version in the URL or to force the clients of your REST API to include a version in the URL they make requests to.

In this lab you will add a new version to the Polls REST API you have been creating to familiarize yourself with the versioning functionality in Scripted REST APIs.

Lab 5 Versioning

BEST PRACTICES

Do: Version your REST API. By default, you do not need to create a version when creating a Scripted REST API in ServiceNow. Best practice is to version your REST API and disable the default route so that consumers must explicitly include the version number in their request URL. In this way you allow clients to decide if and when they want to use a later version of your REST API. If you need to force them to move to a new version at a later point in time you have the ability to disable versions.

Create Lab 5 starting branch

1. In Studio, navigate to Source Control > Create Branch.
2. In the pop-up window, enter a branch name, then select **Lab5-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab5-branch**

Create from Tag: **Lab5-start**

3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab5-branch**.
5. You are now ready to start **Lab 5**.

Add Version to Poll REST API

6. Open Poll API and click Add New version.

The screenshot shows the 'Scripted REST Service' configuration for the 'Poll' API. The 'Name' is set to 'Poll' and 'API ID' to 'poll'. The 'Application' is 'Polls' and the 'API namespace' is 'x_snc_polls'. The 'Base API path' is '/api/x_snc_polls/poll'. The 'Protection policy' is set to '-- None --'. The 'Versioning' tab is active, showing a note about Default ACLs and a list of conditions for enforcement. Below this, there's a 'More info' link. The 'Default ACLs' section shows a lock icon. At the bottom, there are 'Update' and 'Delete' buttons. The 'Related Links' section includes 'Add new version', 'Explore REST API', and 'API analytics', with 'Add new version' being highlighted by a red box.

7. Select version 1 to copy resources. Click Ok.

The screenshot shows the 'Add new version' dialog. It has a checkbox labeled 'Make this version the default' which is not checked. Below it, a dropdown menu labeled 'Copy existing resources from version:' is set to 'v1'. At the bottom, there are 'Cancel' and 'OK' buttons, with the 'OK' button being highlighted by a red box.

To add a new version, use the 'Add new version' link below. You can select one version as the default. Clients can access the default version using either the versioned or non-versioned URI path.

Versions may also be inactivated or deprecated:

- Resources belonging to inactive versions cannot serve requests
- Resources belonging to deprecated versions can serve requests, but are identified as 'Deprecated' in documentation

[More info](#)

	Version ID	Is default	Active	Deprecated
X	v2	false	true	false
X	v1	false	true	false
Insert a new row...				

NOTE: Both versions of the API have default set to **false**. This means that clients consuming this API must include the resource version in the URL.

	Name	HTTP method	Relative path	Resource path	API version	Active
<input type="checkbox"/>	Create new poll	POST	/	/api/x_snc_polls/v1/poll	v1	true
<input type="checkbox"/>	Create new poll (v2)	POST	/	/api/x_snc_polls/v2/poll	v2	true
<input type="checkbox"/>	Delete poll	DELETE	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true
<input type="checkbox"/>	Delete poll (v2)	DELETE	/{poll_id}	/api/x_snc_polls/v2/poll/{poll_id}	v2	true
<input type="checkbox"/>	Edit poll	PATCH	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true
<input type="checkbox"/>	Edit poll (v2)	PATCH	/{poll_id}	/api/x_snc_polls/v2/poll/{poll_id}	v2	true
<input type="checkbox"/>	Retrieve poll details	GET	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true

NOTE: Observe every resource in v1 is copied and added to v2

Test with Postman

8. In Postman, review the requests named ‘CC17: Create new poll v1’ and ‘Create new poll v2’ noting the version number is explicitly specified in the URL for these two requests.
9. Add the tests that you added to the ‘CC17: Create new poll’ request to this the V1 and V2 requests. The behavior between the V1 and V2 resources has not been updated so you can copy and paste your test cases from the ‘CC17: Create new poll’ request and the tests should pass.
10. Verify that your test cases pass successfully.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

11. Similar to creating the Lab2 starting branch, the completed lab can also be checked out from a tag (**Lab5-complete**) in Source control.
12. In **Studio**, navigate to **Source Control > Create Branch**.
13. In the pop-up window, enter a branch name, then select **Lab5-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab5-branch-complete**

Create from Tag: **Lab5-complete**

14. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

15. Verify Studio is on branch **my-Lab5-branch-complete**.

16. Close the REST API Explorer and API Analytics dashboard windows.

Lab 5 is complete. You are now ready to begin lab 6.

Lab Goal

Errors... they happen to the best of us. Whether you are making requests to a 3rd party REST API or your own REST API there are times when you receive errors. Ideally the error message provides you (the client) with enough information to realize what went wrong, if it was your fault (client) or their fault (REST API) and how you can proceed.

Scripted REST APIs provide a helper API (`sn_ws_err`) to make it easier for you as the REST API designer to easily return consistent and informative error messages from your REST API.

Lab 6 Error handling

Create Lab 6 starting branch

1. In **Studio**, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab6-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab6-branch**
Create from Tag: **Lab6-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab6-branch**.
5. You are now ready to start **Lab 6**.

Add Error handling to API – Retrieve poll detail

6. Open Retrieve poll detail (v2).
7. Modify script to check if poll record exists and send a 404 error response.

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab6_retrieve_poll_detail_v2

```
function process( /*RESTAPIrequest*/ request, /*RESTAPIResponse*/ response) {
    var id = request.pathParams.poll_id;
    var pollHelper = new x_snc_polls.PollData_Retriever();
    var pollRecord = new GlideRecordSecure("x_snc_polls_poll");
    pollRecord.get(id);
    if (!pollRecord.isValidRecord()) {
        throw new sn_ws_err.NotFoundError("Poll not found");
    }
    var pollResponse = {
        name: pollRecord.getValue("name"),
        questions: pollHelper.getQuestions(id).as_list,
    };
    return pollResponse;
})(request, response);
```

Test with REST API Explorer

8. Open ‘Retrieve poll detail (v2)’ resource and Click **Explore REST API** in related actions.
9. **Version v2 of Retrieve poll results resource is preselected in API Explorer.**

Namespace: x_snc_polls

API Name: Poll

API Version: v2

Create new poll (v2) (POST)

▶ Retrieve poll details (v2) (GET) **Selected**

Delete poll (v2) (DELETE)

Edit poll (v2) (PATCH)

Retrieve poll results (v2) (GET)

Vote in poll (v2) (POST)

Poll

Retrieve poll details (v2) - Retrieve poll, questions, choices and votes polled by sysid

GET http://10.11.91.87:16001/api/x_snc_polls/v2/poll/{poll_id}

Prepare request

Path parameters

Name	Value
* poll_id	

Query parameters

10. Fill in **invalid** sys_id of poll and click Send.
11. Verify response status code is **404-Not Found**.

Response	
Status code	404 Not Found
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 18:35:47 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
<pre>{ "error": { "detail": "", "message": "Poll not found" }, "status": "failure" }</pre>	

Create tests in Postman

12. In Postman select the ‘CC17: Retrieve poll detail V2’ request’. This is a pre-built request that will make a request to the ‘Retrieve poll detail V2’ resource.
13. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Be sure to specify an **invalid** poll_id. Send a request and verify that you receive a **404 Not Found** status code and that the response body contains the same error message you received in REST API Explorer.
14. Add tests that verify the following details in the response:
 - Response status code is 404
 - Response body contains “error”
 - Response headers include Content-Type of application/json;charset=UTF-8
 - Response body is JSON and contains a status property with a value of ‘failure’.

NOTE: Postman has a test feature that allows you to parse a JSON response and verify it contains a specific property and value. See if you can figure out how to use it. More info can be found at https://www.getpostman.com/docs/testing_examples.

If you are stuck here you can refer to the pre-built request in the Postman collection named “CC17: Retrieve poll detail V2 w/ TEST” to see this request with tests fully specified.

Add Error handling to API – Vote in poll

15. Open Vote in poll(v2).

16. Modify script to check whether poll record exists as well as to see user already voted for the poll.

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab6_vote_in_poll_v2

```
function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var pollId = request.pathParams.poll_id;
    var pollHelper = new x_snc_polls.PollData_Creator();

    // Validate if poll record exists
    var pollRecord = new GlideRecordSecure("x_snc_polls_poll");
    pollRecord.get(pollId);
    if(!pollRecord.isValidRecord()){
        throw new sn_ws_err.NotFoundError("Poll Not found");
    }

    var voteData = request.body.data.votes;

    // Verify if already voted
    var pollResponseRecord = new GlideRecordSecure("x_snc_polls_poll_response");
    pollResponseRecord.addQuery("poll", pollId);
    pollResponseRecord.addQuery("sys_created_by", gs.getUserName());
    pollResponseRecord.query();
    if(pollResponseRecord.next()){
        throw new sn_ws_err.ConflictError("already voted");
    }

    // Record votes
    pollHelper.voteInPoll(voteData, pollId);

    // Set response details
    response.setStatus(201);
}(request, response);
```

Test with REST API Explorer

Open 'Vote in poll (v2)' resource and Click **Explore REST API** in related actions.

17. Version v2 of Vote in poll resource is preselected in API Explorer.

18. Fill in an invalid sys_id of poll and click Send

19. Verify response status code is **404-Not Found**

Response

Status code	404 Not Found
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 18:35:47 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "error": { "detail": "", "message": "Poll not found" }, "status": "failure" }</br >	

20. Now fill in a **valid sys_id** and request body and click Send to vote.

Specify the request body as shown below. Sample script available for you to copy at:
http://bit.ly/CC17_ScriptedRESTAPI_Lab6_vote_in_poll_v2_sample_request

Request Body

Builder Raw

```
{  
  "votes": [  
    {"question_id": "ab4fe4b9443a1200964fac543127a1eb",  
     "vote": "Yellow"  
    }  
  ]  
}
```

21. Verify response status code is **409-Conflict**.

NOTE: Depending on state of instance, you might need to fire the request twice. The first request is a valid vote while the second request results in a conflict response.

Response	
Status code	409 Conflict
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 19:12:31 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "error": { "detail": "", "message": "Already voted" }, "status": "failure" }</br >	

Create tests in Postman

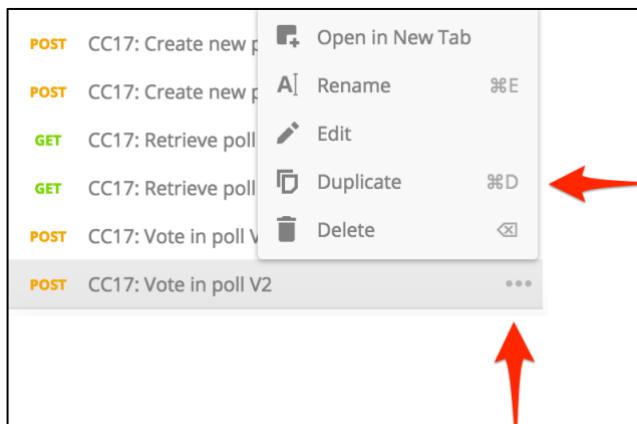
22. In Postman select the 'CC17: Vote in poll V2' request'. This is a pre-built request that will make a request to the 'Vote in poll V2' resource.
23. Update the request replacing the {{instance_url}}, authorization credentials, {{poll_id}}, and body of the request with vote details (question_id and vote value) appropriate for your lab instance.

NOTE: Depending on the state of instance, you might need to fire the request twice. The first request is a valid vote while the second request results in a conflict response. This is because you have added a constraint that users can only vote for a question once.

24. Add tests that verify the following details in the response:

- Response status code is 409 Conflict
- Response body contains ‘error’
- Response headers include Content-Type of ‘application/json; charset=UTF-8’
- Response body is JSON and contains an error property with message property with a value of ‘Already voted’.

25. In Postman create a copy of this request by clicking on the ‘...’ icon to the right of the request name in the left-hand list of requests as shown below and then clicking ‘Duplicate’. This will create a copy of your request that you will update to verify the error message that is returned when you attempt to vote on the same question twice.



26. Update this new request adding tests that verify the following details in the response:

- Response status code is 409
- Response body contains “error”
- Response headers include Content-Type of application/json; charset=UTF-8
- Response body is JSON and contains a status property with a value of ‘failure’.

27. Issue the request and verify that all of your tests have completed successfully.

Add Error handling to API – Retrieve poll results

28. Open Retrieve poll results (v2).
29. Modify script to check if poll exists and send customized error response.

Script: Copy script: http://bit.ly/CC17_ScriptedRESTAPI_Lab6_retrieve_poll_results_v2

```
function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var id = request.pathParams.poll_id;
    var pollHelper = new x_snc_polls.PollDataRetriever();
    var sysId = pollHelper.getSysIdFromGUID(id);
    pollRecord.getSysId();
    if (!pollRecord.isValidRecord()) {
        var err = new ServiceError();
        err.setServiceError();
        err.setStatus(404);
        err.setMessage("Can't find poll with id:" + id);
        err.setDetail("Please provide valid sys_id's required to retrieve results. To get valid sysId, use api/nw/table/x_snc_poll endpoint to get all valid polls");
        throw err;
    }
    var pollResponse = {
        name: pollRecord.getValue("name"),
        questions: pollHelper.getResultsByUser(id)
    };
    return pollResponse;
})(request, response);
```

Test with REST API Explorer

30. Open 'Retrieve poll results (v2)' resource and Click **Explore REST API** in related actions.
31. Version v2 of Retrieve poll results resource is preselected in API Explorer.

REST API Explorer

Namespace: x_snc_polls

API Name: Poll

API Version: v2

Create new poll (v2) (POST)

Retrieve poll details (v2) (GET)

Delete poll (v2) (DELETE)

Edit poll (v2) (PATCH)

► Retrieve poll results (v2) (GET)

vote in poll (v2) (POST)

Poll

Retrieve poll results (v2)

GET http://10.11.91.87:16001/api/x_snc_polls/v2/poll/{poll_id}/results

Prepare request

Path parameters

Name	Value
* poll_id	invalid

32. Fill in invalid sys_id of poll and click Send.

33. Verify response status code is **404-Not Found** with custom Error message.

Response

Status code **404 Not Found**

Headers

Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 19:35:33 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true

Response Body

```
{  
  "error": {  
    "detail": "Valid sysId of record is required to retrieve results. To get valid sysId, use api/now/table/x_snc_polls_poll endpoint to ge  
    "message": "Can't find poll with id:invalid"  
  },  
  "status": "failure"  
}
```

Create tests in Postman

34. Now that you have experience using Postman to build requests and create tests you are on your own for this last Postman test.
35. Create a new Postman request, either brand new or by duplicating an existing request, that makes a request to the **Retrieve poll results V2 resource** with an **invalid** poll id.
36. Add tests to this request verifying the following:
 - Response status code is 404
 - Response body contains “error”
 - Response headers include Content-Type of ‘application/json;charset=UTF-8’
 - Response body is JSON and contains an error property with with a message child property that has a value of ‘Can’t find poll with id:invalid’.
37. Issue request and verify that all tests pass successfully.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

38. Similar to creating the Lab6 starting branch, the completed lab can also be checked out from a tag (**Lab6-complete**) in Source control.
39. In **Studio**, navigate to **Source Control > Create Branch**.
40. In the pop-up window, enter a branch name, then select **Lab6-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab6-branch-complete**

Create from Tag: **Lab6-complete**

41. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

42. Verify Studio is on branch **my-Lab6-branch-complete**.

43. Close the REST API Explorer and API Analytics dashboard windows.

Lab 6 is complete. You are now ready to begin lab 7.

Lab Goal

Congratulations you've made it to the challenge lab. If you've made it this far you have either whizzed right through labs 1-6 and are a Scripted REST API and Postman expert or you're working on this challenge lab after the CreatorCon Workshop.

Throughout this workshop you have built a Scripted REST API for the Polls application and created requests with tests in the Postman collection that allow you to quickly issue requests to your REST API that also have test cases that verify the REST API is returning valid responses and behaving as intended.

Creating tests that validate your REST APIs behavior is a **BEST PRACTICE** that allows you to easily verify your REST API is behaving as intended and quickly identify any breaking changes in your REST API in the future as you make changes to add functionality or patch bugs.

Postman makes it easy to run these tests manually but **Nobody** enjoys running tests manually all the time. Wouldn't it be nice if you could automate these tests? I certainly think so. Let's take this one step further and see if you can get these tests to run from the command line. Postman provides a tool called **Newman** that allows you to run the tests in your existing Postman collection from the command line and see the results either at the command line or write them to a file. For this challenge lab your task is to install Newman and run your Postman collection from the command line. Check out the links below to download and install Newman and get more information on how to run a collection from the command line.

If you can get Newman running from the command line you are only a few steps away (bash, cron, PowerShell script) from automating your REST API test cases with Postman and Newman. This is outside the scope of this lab but you can imagine how you could integrate this with a continuous integration testing tool to have these tests run on a regular basis or as part of a build process.

Newman: <http://blog.getpostman.com/2014/05/12/meet-newman-a-command-line-companion-for-postman/>

Running collections from the command line:
https://www.getpostman.com/docs/newman_intro

Integrating automated API tests with Jenkins:
https://www.getpostman.com/docs/integrating_with_jenkins

Lab 7 Challenge Lab