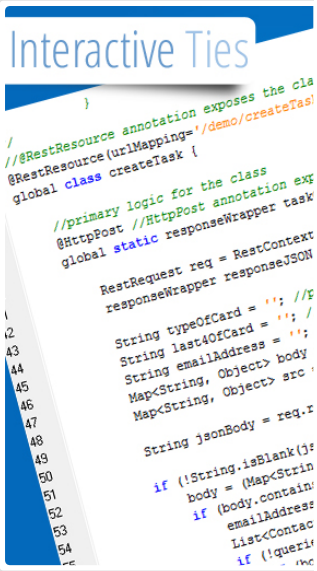
Custom REST API Endpoint for Salesforce



Custom REST API Endpoint for Salesforce

17 Feb

17 February 2017 (../../blog/2017/) By Greg Hacic (../../greg-hacic.php)

REST API (../../blog/listing.php?q=REST+API), JSON (../../blog/listing.php?q=JSON), Wrapper Class (../../blog/listing.php?q=Wrapper+Class), Apex Class (../../blog/listing.php?q=Apex+Class)

Many third party applications have their own APIs or webhooks that allow for easy and usually simplified integrations between platforms. Another integration choice might be to use some type of middleware or enterprise service bus (ESB) to move data around. Either way you can use the Salesforce REST API to do all of the standard CRUD (create, read, update and delete) operations or you can build some custom Apex logic and expose that through the REST API in order to streamline the processing.

For example, let's consider integrating Stripe and Salesforce. Stripe is great at processing credit cards and managing subscription & payment details. Salesforce is great at managing all of the interactions between your business and your customers. So it would be perfectly reasonable to have Stripe inform Salesforce when a payment has been made so that your team is aware.

Let's begin with one of the concepts that will be needed before moving further. A webhook is an HTTP callback that fires when an event happens. An application or platform implementing webhooks will POST a message to a URL when certain things happen. For the purposes of this example, Stripe will POST a JSON message to Salesforce and Salesforce will consume that message and do some additional processing based upon the JSON content.

If you've done any development in the past with web services then you know there is a whole bunch of "stuff" that can be handled through the Salesforce REST API without any custom Apex classes. The API can be used to query for records, insert new records, update records, delete records, etc. For our purposes and to better illustrate the power of the API, we will create a custom Apex class to perform multiple operations.

Let's dive into the example. For the purposes of simplicity we will assume that the JSON POSTed to our Salesforce endpoint will be as follows:

```
    {
        "id": "ch_abcdefghijklmnopqrstuvwx",
        "amount": 9900,
        "description": "greg@interactiveties.com",
        "source": {
            "id": "card_19mlu82GgpHzUJkgFLGaSS8e",
            "brand": "American Express",
            "exp_month": 11,
            "exp_year": 2020,
            "last4": "2222",
            "name": "greg@interactiveties.com",
        },
        "status": "succeeded",
        "type": "charge.succeeded"
    }
```

We need to convert this JSON request into an object that we can use in our Apex class. Then we will query for a Contact with the email address that is identified in the value for the description. In this case, that email address is greg@interactiveties.com. When we find the contact we will use some of the other attributes found in the JSON object in order to create an appropriate Task record associated to the Contact we found in our query.

In response to the request we will provide the Id of the Task that we created and a success message. The JSON we return will look like this:

```
    {
        "status": "success",
        "contactid": "0033000000YVLQWAA5",
        "taskid": "00T3000000rEvJtEAK",
        "message": ""
    }
```

Now that you understand what we're trying to do, let's get to the Apex class:

```
/*
    Created by: Greg Hacic
    Last Update: 17 February 2017 by Greg Hacic
    Questions?: greg@ities.co

    Notes:
        - API endpoint accepts JSON similar to:
            {
                "id": "ch_abcdefghijklmnopqrstuvwx",
                "amount": 9900,
                "description": "greg@interactiveties.com",
                "source": {
                    "id": "card_12345678912345678912345 6",
                    "brand": "American Express",
                    "exp_month": 11,
                    "exp_year": 2021,
                    "last4": "2222",
                    "name": "greg@interactiveties.com"
                },
                "status": "succeeded",
                "type": "charge.succeeded"
            }
        - queries for the Contact with the email address provided in the description key/value pair from the JSON request
        - creates a Task
        - returns JSON similar to:
            {
                "status": "success",
                "contactid": "0033000000YVLQWAA5",
                "taskid": "00T3000000rEvJtEAK",
                "message": ""
            }
*/
//@RestResource annotation exposes the class as a REST resource
@RestResource(urlMapping='/demo/createTask/*') //endpoint definition > {Salesforce Base URL}/services/apexrest/demo/createTask/
global class createTask {

    //primary logic for the class
    @HttpPost //HttpPost annotation exposes the method as a REST resource and called when an HTTP POST request is sent
    global static responseWrapper taskCreationLogic() {

        RestRequest req = RestContext.request; //the RestRequest for the Apex REST method
        responseWrapper responseJSON = new responseWrapper(); //responseWrapper object for API response

        String typeOfCard = ''; //placeholder for the type of card string
        String last4OfCard = ''; //placeholder for the last four digits of the card
        String emailAddress = ''; //placeholder for an email address
        Map<String, Object> body = new Map<String, Object>(); //placeholder for the JSON Body of the request
        Map<String, Object> src = new Map<String, Object>(); //placeholder for the source object from the JSON request

        String jsonBody = req.requestBody.toString(); //the body of the request

        if (!String.isBlank(jsonBody)) { //if the request body is NOT white space, empty ('') or null
            body = (Map<String, Object>)JSON.deserializeUntyped(jsonBody); //deserializes the JSON string into collections of pri
            if (body.containsKey('description')) { //if there is a key of description in our body map
                emailAddress = (String)body.get('description'); //grab the value for the description key from the body map and co
                List<Contact> queriedContacts = [SELECT Id FROM Contact WHERE Email = :emailAddress ORDER BY CreatedDate DESC LIM
                if (!queriedContacts.isEmpty()) { //if the list is not empty
                    if (body.containsKey('source')) { //if there is a key of source in our body map
                        src = (Map<String, Object>)body.get('source'); //grab the value for the source key from the body map and
                        if (src.containsKey('brand')) { //if there is a key of brand in our src map
                            typeOfCard = (String)src.get('brand'); //grab the value for the brand key from the src map and cast i
                        }
                        if (src.containsKey('last4')) { //if there is a key of last4 in our src map
                            last4OfCard = (String)src.get('last4'); //grab the value for the last4 key from the src map and cast
                        }
                    }

                    responseJSON.contactid = queriedContacts[0].Id; //populate the Id of the Contact record to our response objec

                    Task newTask = new Task(ActivityDate = Date.Today(), Description = 'The '+typeOfCard+' credit card ending in

                    Database.SaveResult insertNewTask = Database.insert(newTask); //insert the new Task
                    if (!insertNewTask.isSuccess()) { //if the insert DML was NOT successful
                        List<Database.Error> errors = insertNewTask.getErrors(); //grab the error array from the SaveResult objec
                        //respond with failure
                        responseJSON.status = 'failure';
                        responseJSON.message = errors[0].getMessage(); //set the message to the first error in the array
                    } else { //otherwise, the insert was successful
                        responseJSON.taskid = insertNewTask.getId(); //populate the Id of the Task record to our response object
                    }
                } else { //otherwise, no key of source in our map
                    //respond with failure
                    responseJSON.status = 'failure';
                    responseJSON.message = 'There are no Contacts with the email address of '+emailAddress+'.';
                }
            } else { //otherwise, no key of description in our map
```

```
87                  //respond with failure
88                  responseJSON.status = 'failure';
89                  responseJSON.message = 'No description in the JSON request.';
90              }
91          } else { //otherwise, the JSON body was white space, empty ('') or null
92              //respond with failure
93              responseJSON.status = 'failure';
94              responseJSON.message = 'Things basically broke...';
95          }
96          return responseJSON; //return the JSON response
97      }
98
99      //wrapper class for the response to an API request
100     global class responseWrapper {
101
102         global String status {get;set;} //status string
103         global String contactid {get;set;} //18 character Contact record Id
104         global String taskid {get;set;} //18 character Task record Id
105         global String message {get;set;} //message string
106
107         //constructor
108         global responseWrapper() {
109             //default all values
110             this.status = 'success';
111             this.contactid = '';
112             this.taskid = '';
113             this.message = '';
114         }
115     }
116
117 }
```

The purpose of this post is to provide you with an example that you can use in combination with the samples that exist in the developer community in order to better familiarize you with the core concepts and options that are available for creating REST API Web Services with Salesforce.

Author


(../../greg-hacic.php)

Greg Hacic (../../greg-hacic.php) in (https://www.linkedin.com/in/greghacic/) (http://www.twitter.com/greghacic)
I've been working with Salesforce since 2003. Over the years I've held various roles for diverse salesforce.com customers, created a Salesforce specific ISV, founded a few start-ups and built numerous applications for the AppExchange. All of these experiences have allowed me to learn quite a bit about building on the platform.

Categories

- Content Tag
- Calendar Year

★ Popular | Recent

Custom REST API Endpoint for Salesforce (../../blog/2017/custom-rest-api-endpoint.php)
17 Feb 2017

Wrapper Class Example for Visualforce Page (../../blog/2012/visualforce-wrapper-class.php)
1 Jun 2012

Interactive Ties (../../index.php)

Custom Salesforce Button to Execute JavaScript (../../blog/2010/execute-javascript-button.php)
14 Jul 2010

@testSetup Annotation Sample (../../blog/2015/apex-testsetup-annotation.php)
31 Jul 2015

About

This is an educational blog for people interested in development on the Salesforce platform.

Our goal is to share information about how we build on the platform so you can gain insight into what may or may not work for you.

Suggestions

Have a topic or concept that you would like us to write about? Send us your questions, topics, ideas or feedback. ideas@ities.co (mailto:ideas@ities.co)