Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2019, 6(6):210-217

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Guide to Implement a REST API Framework in Salesforce

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services
Phoenix, Arizona, USA
ChiragPethad@live.com, ChiragPethad@gmail.com, Cpethad@petsmart.com

Abstract: This document provides a comprehensive guide to implementing a REST API framework in Salesforce Apex. It discusses the key features of a REST API framework, provides code examples for error handling, inputs and outputs, and dispatcher and controller components. The document also highlights best practices for implementing a REST API framework and concludes by emphasizing the importance of a well-designed framework for seamless integration and improved efficiency in Salesforce applications.

Keywords: REST, API, Request, Response, Framework, Programming, Apex, Error Handling.

Introduction

In today's interconnected digital landscape, APIs (Application Programming Interfaces) play a critical role in enabling communication between different systems and applications. Salesforce, a leading customer relationship management (CRM) platform, provides robust support for building and consuming REST APIs through its Apex programming language. This white paper explores the principles, benefits, and best practices for implementing a REST API framework in Salesforce Apex.

The Importance of Rest APIS

REST (Representational State Transfer) APIs are a popular architectural style for designing networked applications. They are lightweight, easy to use, and well-suited for web and mobile applications. Key benefits of REST APIs include:

A. **Interoperability**: REST APIs enable seamless communication between different systems, regardless of the underlying technology stack.

B. **Scalability**: RESTful services can handle a large number of requests and are easily scalable.

C. **Flexibility:** REST APIs support a wide range of data formats (e.g., JSON, XML), making them versatile and adaptable to different use cases.

D. **Simplicity:** REST APIs use standard HTTP methods (GET, POST, PUT, DELETE), making them easy to understand and implement.

Key Features of Rest Api Framework

An effective REST API framework in Salesforce Apex should include the following features:

A. **CRUD Operations**: Support for creating, reading, updating, and deleting records.

B. **Error Handling:** Comprehensive error handling to provide meaningful feedback to API consumers.

C. **Documentation:** Clear and concise documentation to help developers understand and use the API effectively.

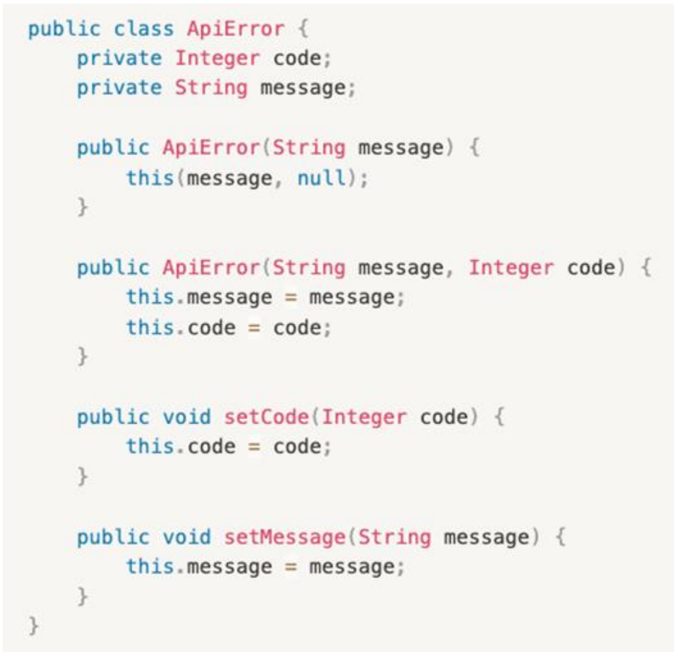
D. **Code Best Practices:** Reduce the need for boiler plate code for creating a new API.

E. **Development:** Reduce Development and Testing time and effort.

Implementing a Rest API Framework

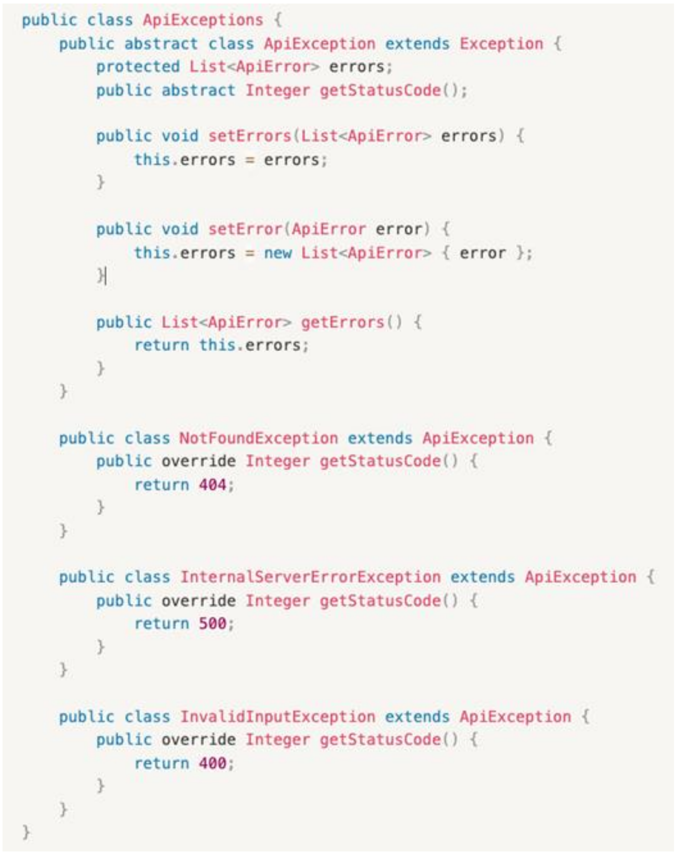
A. Error Handling

We start by creating the components required for an efficient, consistent error handling.



```
public class ApiError {
    private Integer code;
    private String message;

    public ApiError(String message) {
        this(message, null);
    }

    public ApiError(String message, Integer code) {
        this.message = message;
        this.code = code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

We also create an apex class for collection of all API related exceptions. It consists of a top level class that will be used to define all inner classes for handling specific errors like Bad / Invalid Inputs (400), Not Found (404) and Internal Server Errors (500).

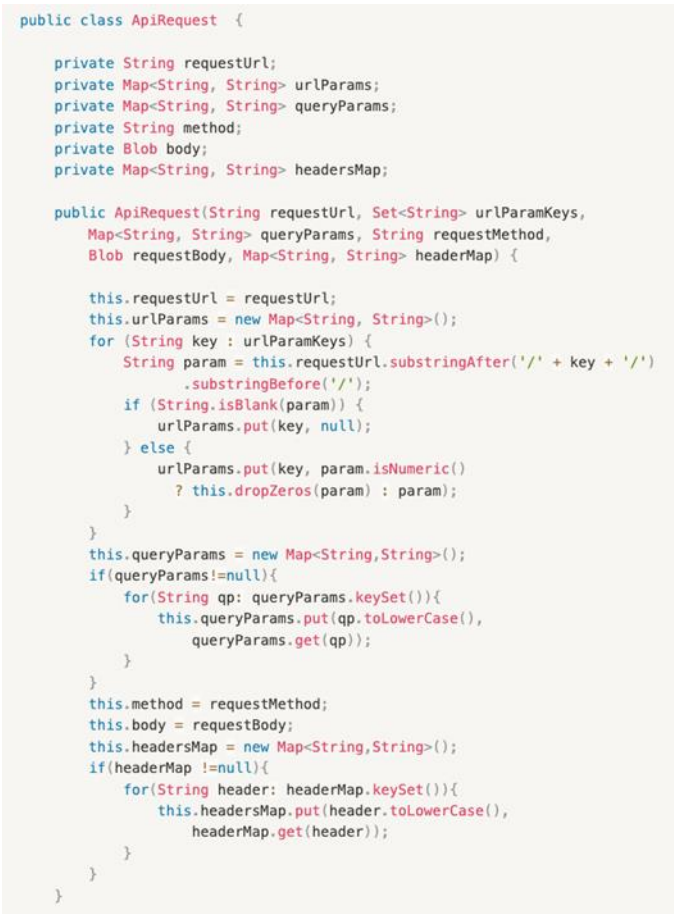


```
public class ApiExceptions {
    public abstract class ApiException extends Exception {
        protected List<ApiError> errors;
        public abstract Integer getStatusCode();

        public void setErrors(List<ApiError> errors) {
            this.errors = errors;
        }

        public void setError(ApiError error) {
            this.errors = new List<ApiError> { error };
        }

        public List<ApiError> getErrors() {
            return this.errors;
        }
    }

    public class NotFoundException extends ApiException {
        public override Integer getStatusCode() {
            return 404;
        }
    }

    public class InternalServerErrorException extends ApiException {
        public override Integer getStatusCode() {
            return 500;
        }
    }

    public class InvalidInputException extends ApiException {
        public override Integer getStatusCode() {
            return 400;
        }
    }
}
```

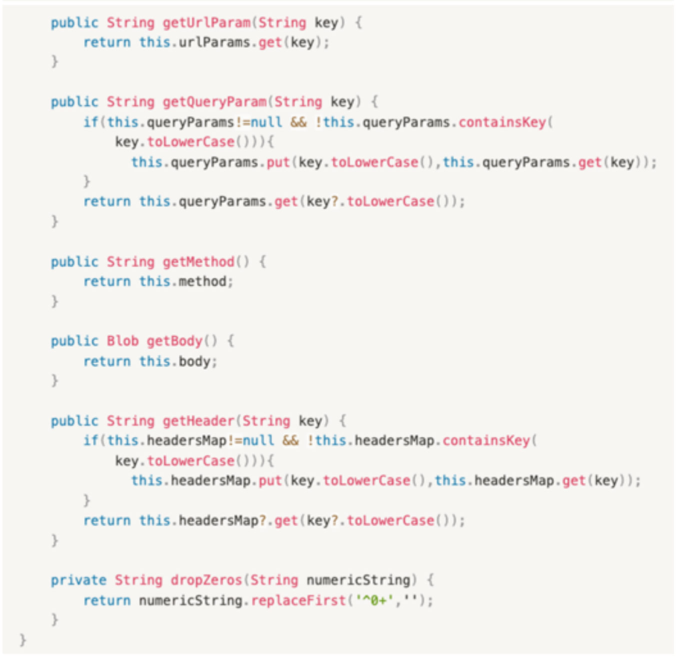
B. Request and Response

Next, we create boiler plate code for Inputs and Outputs or Requests and Responses.

Api Request Class represents the request attribute from the Rest Context. We parse the request URL to retrieve all the URL parameters and drop any zeros from the values if the values are numeric. We also retrieve any query parameters from the request URL. As well as we capture the header parameters.¬ Framework will be responsible to instantiate the Api request with all the attributes from the REST Context.

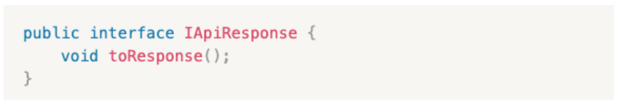


```
public class ApiRequest  {

    private String requestUrl;
    private Map<String, String> urlParams;
    private Map<String, String> queryParams;
    private String method;
    private Blob body;
    private Map<String, String> headersMap;

    public ApiRequest(String requestUrl, Set<String> urlParamKeys,
        Map<String, String> queryParams, String requestMethod,
        Blob requestBody, Map<String, String> headerMap) {

        this.requestUrl = requestUrl;
        this.urlParams = new Map<String, String>();
        for (String key : urlParamKeys) {
            String param = this.requestUrl.substringAfter('/' + key + '/')
                  .substringBefore('/');
            if (String.isBlank(param)) {
                urlParams.put(key, null);
            } else {
                urlParams.put(key, param.isNumeric()
                  ? this.dropZeros(param) : param);
            }
        }
        this.queryParams = new Map<String,String>();
        if(queryParams!=null){
            for(String qp: queryParams.keySet()){
                this.queryParams.put(qp.toLowerCase(),
                    queryParams.get(qp));
            }
        }
        this.method = requestMethod;
        this.body = requestBody;
        this.headersMap = new Map<String,String>();
        if(headerMap !=null){
            for(String header: headerMap.keySet()){
                this.headersMap.put(header.toLowerCase(),
                    headerMap.get(header));
            }
        }
    }
```

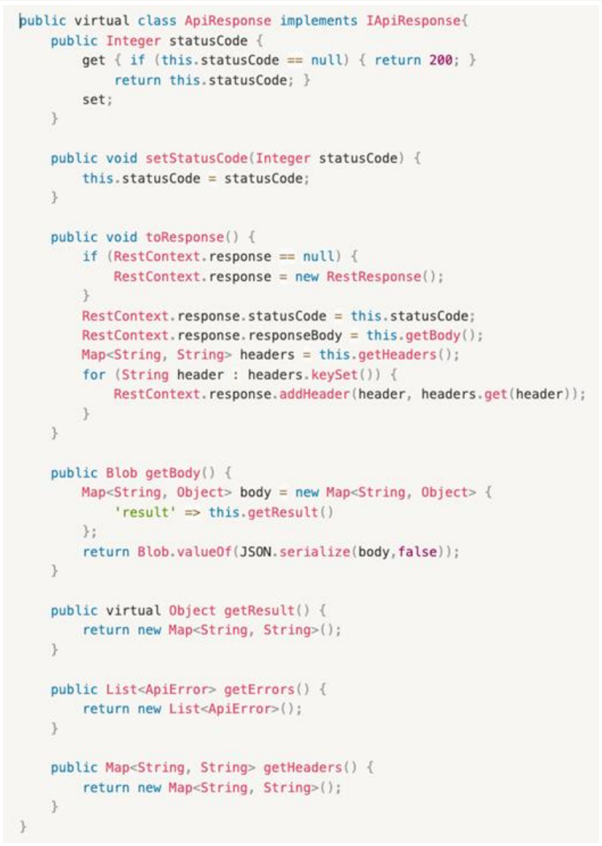


```
    public String getUrlParam(String key) {
        return this.urlParams.get(key);
    }

    public String getQueryParam(String key) {
        if(this.queryParams!=null && !this.queryParams.containsKey(
            key.toLowerCase())){
              this.queryParams.put(key.toLowerCase(),this.queryParams.get(key));
        }
        return this.queryParams.get(key?.toLowerCase());
    }

    public String getMethod() {
        return this.method;
    }

    public Blob getBody() {
        return this.body;
    }

    public String getHeader(String key) {
        if(this.headersMap!=null && !this.headersMap.containsKey(
            key.toLowerCase())){
              this.headersMap.put(key.toLowerCase(),this.headersMap.get(key));
        }
        return this.headersMap?.get(key?.toLowerCase());
    }

    private String dropZeros(String numericString) {
        return numericString.replaceFirst('^0+','');
    }
}
```

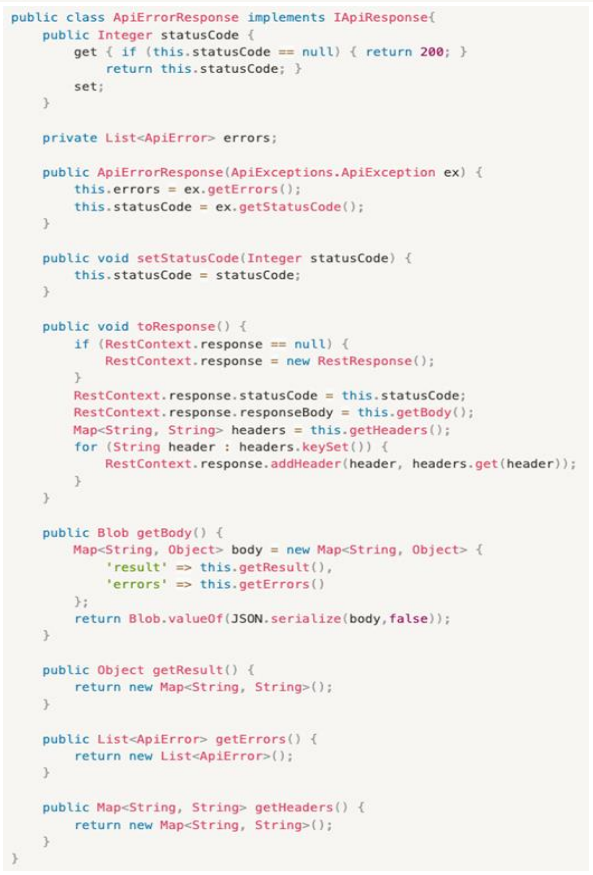
We also create an Interface for Response object and two implementations - One for Api Response (Success) and other for Errors.



```
public interface IApiResponse {
    void toResponse();
}
```



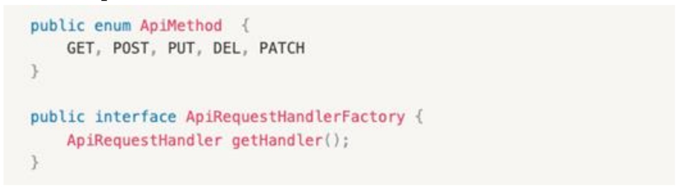
```
public virtual class ApiResponse implements IApiResponse{
    public Integer statusCode {
        get { if (this.statusCode == null) { return 200; }
            return this.statusCode; }
        set;
    }

    public void setStatusCode(Integer statusCode) {
        this.statusCode = statusCode;
    }

    public void toResponse() {
        if (RestContext.response == null) {
            RestContext.response = new RestResponse();
        }
        RestContext.response.statusCode = this.statusCode;
        RestContext.response.responseBody = this.getBody();
        Map<String, String> headers = this.getHeaders();
        for (String header : headers.keySet()) {
            RestContext.response.addHeader(header, headers.get(header));
        }
    }

    public Blob getBody() {
        Map<String, Object> body = new Map<String, Object> {
            'result' => this.getResult()
        };
        return Blob.valueOf(JSON.serialize(body,false));
    }

    public virtual Object getResult() {
        return new Map<String, String>();
    }

    public List<ApiError> getErrors() {
        return new List<ApiError>();
    }

    public Map<String, String> getHeaders() {
        return new Map<String, String>();
    }
}
```



```
public class ApiErrorResponse implements IApiResponse{
    public Integer statusCode {
        get { if (this.statusCode == null) { return 200; }
            return this.statusCode; }
        set;
    }

    private List<ApiError> errors;

    public ApiErrorResponse(ApiExceptions.ApiException ex) {
        this.errors = ex.getErrors();
        this.statusCode = ex.getStatusCode();
    }

    public void setStatusCode(Integer statusCode) {
        this.statusCode = statusCode;
    }

    public void toResponse() {
        if (RestContext.response == null) {
            RestContext.response = new RestResponse();
        }
        RestContext.response.statusCode = this.statusCode;
        RestContext.response.responseBody = this.getBody();
        Map<String, String> headers = this.getHeaders();
        for (String header : headers.keySet()) {
            RestContext.response.addHeader(header, headers.get(header));
        }
    }

    public Blob getBody() {
        Map<String, Object> body = new Map<String, Object> {
            'result' => this.getResult(),
            'errors' => this.getErrors()
        };
        return Blob.valueOf(JSON.serialize(body,false));
    }

    public Object getResult() {
        return new Map<String, String>();
    }

    public List<ApiError> getErrors() {
        return new List<ApiError>();
    }

    public Map<String, String> getHeaders() {
        return new Map<String, String>();
    }
}
```

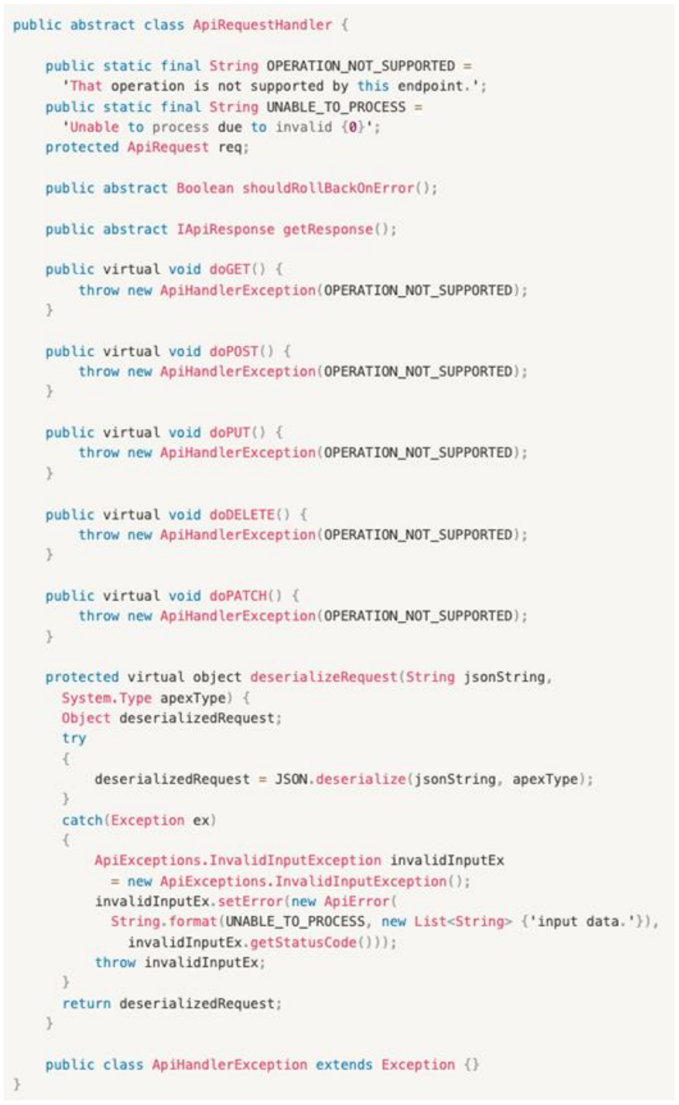
C. Dispatcher and Controller

Next we a Base Factory class and a Base Handler class that needs to be extended by the Rest APIs. These classes provides the default implementation for the boiler plate code. We start by defining an ENUM with the REST verbs we are going to support. Next, we define the Interface for Api Request Handler Factory. The implementer of this interface will provide the definition to instantiate an instance of the Api Request Handler.



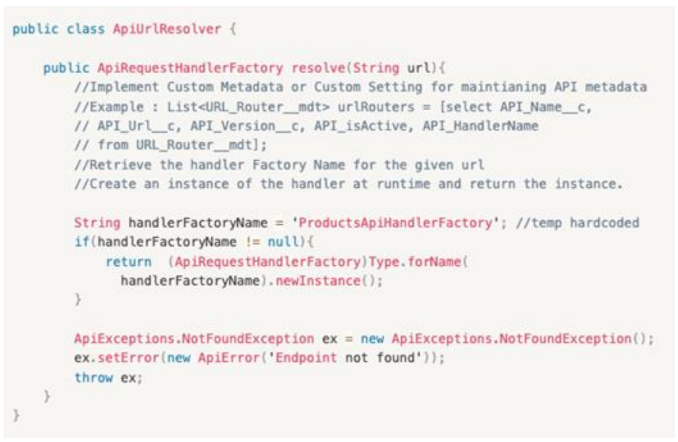
```
public enum ApiMethod  {
    GET, POST, PUT, DEL, PATCH
}

public interface ApiRequestHandlerFactory {
    ApiRequestHandler getHandler();
}
```

We will then create an Abstract or Base implementation of Api Request Handler Class which provides the default implementation for all the HTTP Verbs we support. This helps the actual handler to avoid implementing HTTP methods the service do not support. The child class is only responsible for overriding the HTTP Method that meets business requirements.

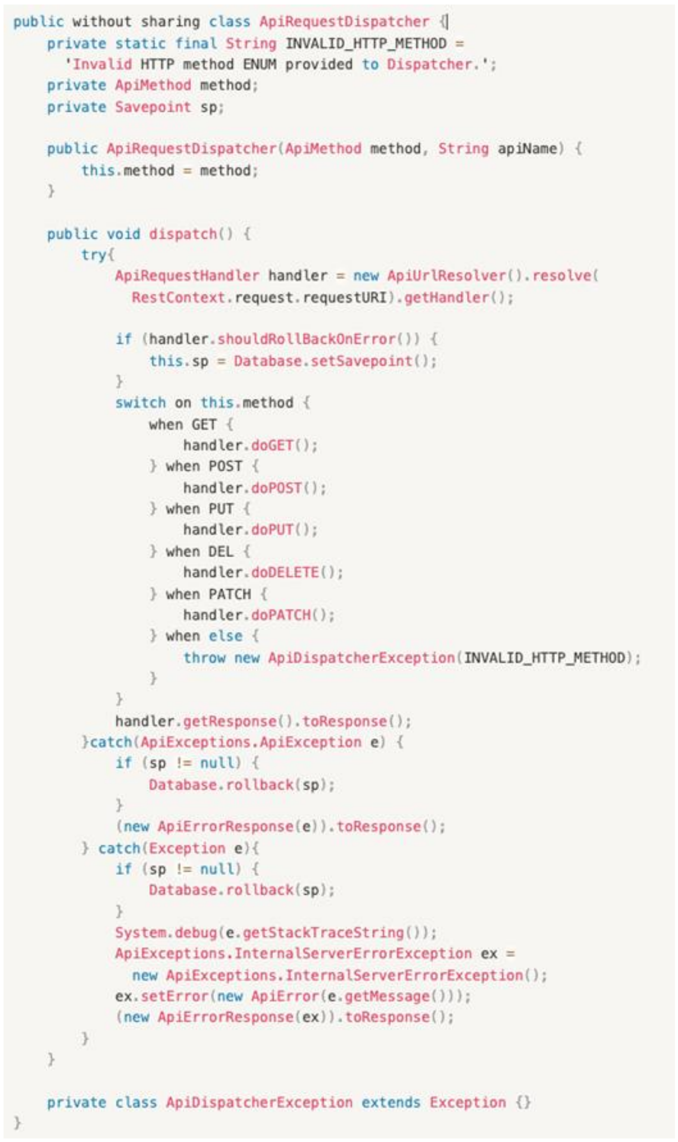


```
public abstract class ApiRequestHandler {

    public static final String OPERATION_NOT_SUPPORTED =
      'That operation is not supported by this endpoint.';
    public static final String UNABLE_TO_PROCESS =
      'Unable to process due to invalid {0}';
    protected ApiRequest req;

    public abstract Boolean shouldRollBackOnError();

    public abstract IApiResponse getResponse();

    public virtual void doGET() {
        throw new ApiHandlerException(OPERATION_NOT_SUPPORTED);
    }

    public virtual void doPOST() {
        throw new ApiHandlerException(OPERATION_NOT_SUPPORTED);
    }

    public virtual void doPUT() {
        throw new ApiHandlerException(OPERATION_NOT_SUPPORTED);
    }

    public virtual void doDELETE() {
        throw new ApiHandlerException(OPERATION_NOT_SUPPORTED);
    }

    public virtual void doPATCH() {
        throw new ApiHandlerException(OPERATION_NOT_SUPPORTED);
    }

    protected virtual object deserializeRequest(String jsonString,
      System.Type apexType) {
      Object deserializedRequest;
      try
      {
          deserializedRequest = JSON.deserialize(jsonString, apexType);
      }
      catch(Exception ex)
      {
          ApiExceptions.InvalidInputException invalidInputEx
            = new ApiExceptions.InvalidInputException();
          invalidInputEx.setError(new ApiError(
            String.format(UNABLE_TO_PROCESS, new List<String> {'input data.'}),
              invalidInputEx.getStatusCode()));
          throw invalidInputEx;
      }
      return deserializedRequest;
    }

    public class ApiHandlerException extends Exception {}
}
```

The final step in the framework is to create Dispatcher component. This centralized API dispatcher provides a mechanism for keeping API request handling DRY Leverages the Api Request Handler abstract class the Api Method ENUM for operations. Web services should instantiate this class with the appropriate handler and method. e.g. a sample GET web service would inject a Account Handler and Api Method GET.



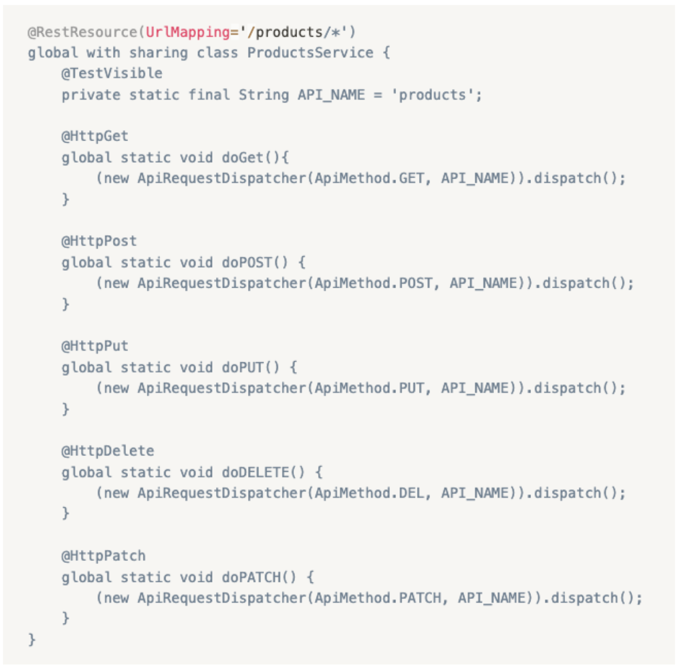
```
public class ApiUrlResolver {

    public ApiRequestHandlerFactory resolve(String url){
        //Implement Custom Metadata or Custom Setting for maintianing API metadata
        //Example : List<URL_Router__mdt> urlRouters = [select API_Name__c,
        // API_Url__c, API_Version__c, API_isActive, API_HandlerName
        // from URL_Router__mdt];
        //Retrieve the handler Factory Name for the given url
        //Create an instance of the handler at runtime and return the instance.

        String handlerFactoryName = 'ProductsApiHandlerFactory'; //temp hardcoded
        if(handlerFactoryName != null){
            return  (ApiRequestHandlerFactory)Type.forName(
              handlerFactoryName).newInstance();
        }

        ApiExceptions.NotFoundException ex = new ApiExceptions.NotFoundException();
        ex.setError(new ApiError('Endpoint not found'));
        throw ex;
    }
}
```



```
public without sharing class ApiRequestDispatcher {
    private static final String INVALID_HTTP_METHOD =
      'Invalid HTTP method ENUM provided to Dispatcher.';
    private ApiMethod method;
    private Savepoint sp;

    public ApiRequestDispatcher(ApiMethod method, String apiName) {
        this.method = method;
    }

    public void dispatch() {
        try{
            ApiRequestHandler handler = new ApiUrlResolver().resolve(
              RestContext.request.requestURI).getHandler();

            if (handler.shouldRollBackOnError()) {
                this.sp = Database.setSavepoint();
            }
            switch on this.method {
                when GET {
                    handler.doGET();
                } when POST {
                    handler.doPOST();
                } when PUT {
                    handler.doPUT();
                } when DEL {
                    handler.doDELETE();
                } when PATCH {
                    handler.doPATCH();
                } when else {
                    throw new ApiDispatcherException(INVALID_HTTP_METHOD);
                }
            }
            handler.getResponse().toResponse();
        }catch(ApiExceptions.ApiException e) {
            if (sp != null) {
                Database.rollback(sp);
            }
            (new ApiErrorResponse(e)).toResponse();
        } catch(Exception e){
            if (sp != null) {
                Database.rollback(sp);
            }
            System.debug(e.getStackTraceString());
            ApiExceptions.InternalServerErrorException ex =
              new ApiExceptions.InternalServerErrorException();
            ex.setError(new ApiError(e.getMessage()));
            (new ApiErrorResponse(ex)).toResponse();
        }
    }

    private class ApiDispatcherException extends Exception {}
}
```

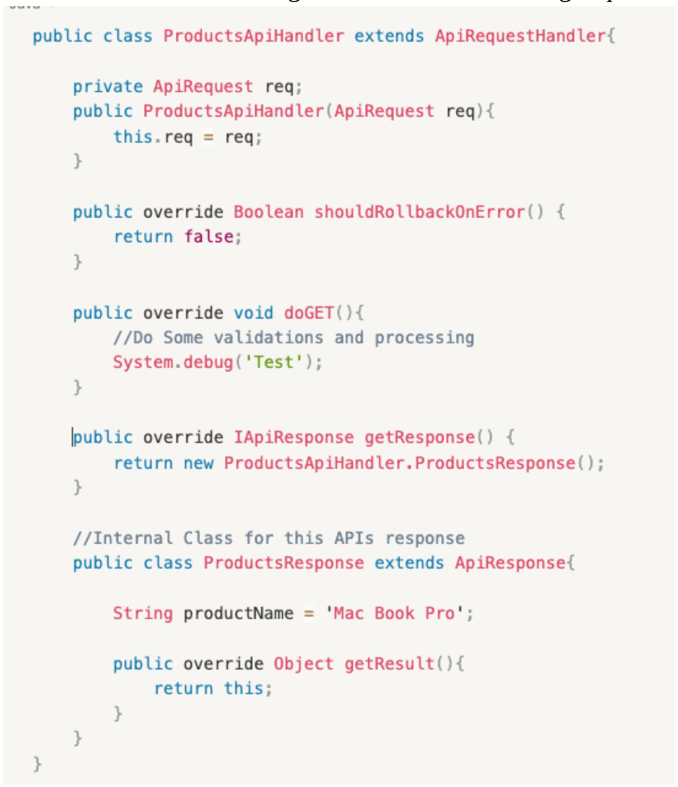
D. API Implementation Example using the framework

Now with the Framework Implementation is done. Lets see how to implement a new REST API using the framework we just created. We start by defining the web service class as follows:



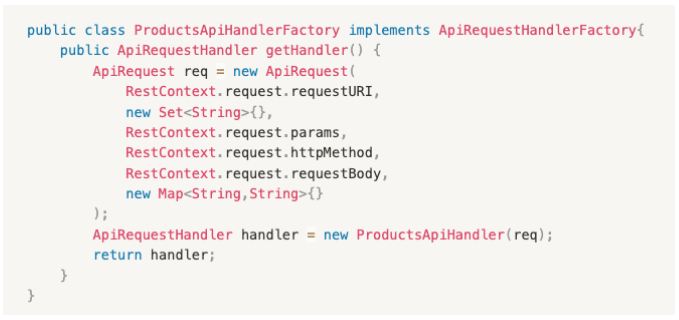
```
@RestResource(UrlMapping='/products/*')
global with sharing class ProductsService {
    @TestVisible
    private static final String API_NAME = 'products';

    @HttpGet
    global static void doGet(){
        (new ApiRequestDispatcher(ApiMethod.GET, API_NAME)).dispatch();
    }

    @HttpPost
    global static void doPOST() {
        (new ApiRequestDispatcher(ApiMethod.POST, API_NAME)).dispatch();
    }

    @HttpPut
    global static void doPUT() {
        (new ApiRequestDispatcher(ApiMethod.PUT, API_NAME)).dispatch();
    }

    @HttpDelete
    global static void doDELETE() {
        (new ApiRequestDispatcher(ApiMethod.DEL, API_NAME)).dispatch();
    }

    @HttpPatch
    global static void doPATCH() {
        (new ApiRequestDispatcher(ApiMethod.PATCH, API_NAME)).dispatch();
    }
}
```

Next we create the API Handler class that extends the Base API Handler Class that is part of the framework. This class is used to implement the core business logic to handle the incoming requests and respond to it.



```
public class ProductsApiHandler extends ApiRequestHandler{

    private ApiRequest req;
    public ProductsApiHandler(ApiRequest req){
        this.req = req;
    }

    public override Boolean shouldRollbackOnError() {
        return false;
    }

    public override void doGET(){
        //Do Some validations and processing
        System.debug('Test');
    }

    public override IApiResponse getResponse() {
        return new ProductsApiHandler.ProductsResponse();
    }

    //Internal Class for this APIs response
    public class ProductsResponse extends ApiResponse{

        String productName = 'Mac Book Pro';

        public override Object getResult(){
            return this;
        }
    }
}
```

And lastly we will tell the framework which API handler to invoke for this API / URL by implementing the Factory class.



```
public class ProductsApiHandlerFactory implements ApiRequestHandlerFactory{
    public ApiRequestHandler getHandler() {
        ApiRequest req = new ApiRequest(
            RestContext.request.requestURI,
            new Set<String>{},
            RestContext.request.params,
            RestContext.request.httpMethod,
            RestContext.request.requestBody,
            new Map<String,String>{}
        );
        ApiRequestHandler handler = new ProductsApiHandler(req);
        return handler;
    }
}
```

Best Practices for Rest API Framework

A. **Use Standard HTTP Methods**: Adhere to standard HTTP methods (GET, POST, PUT, DELETE) to ensure consistency and predictability.

B. **Use Standardized Errors and Responses:** Ensure there is consistency in how the framework return the Error or Success response to the consumer of the API.

C. **Implement Rollback options:** Implement automatic roll backs when there is any error in processing the request to ensure data is in pristine state.

D. **Focus on Business Logic**: Framework should be doing all the heavy lifting and boiler plate code so that developers can focus on the core business logic.

Conclusion

Implementing a REST API framework in Salesforce Apex is essential for enabling seamless integration with other systems and applications. By following the principles, best practices, and implementation steps outlined in this white paper, developers can create robust, scalable, and secure REST APIs that enhance the capabilities of their Salesforce applications. With a well-designed REST API framework, businesses can improve interoperability, scalability, and flexibility, driving greater efficiency and innovation in their operations.

References

[1]. REST API- https://developer.salesforce.com/docs/atlas.en us.api_rest.meta/api_rest/intro_what_is_rest_api.htm
[2]. Salesforce REST API Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_rest.htm
[3]. https://www.salesforceben.com/exploring-the-salesforce-rest-api/
[4]. https://www.apexhours.com/rest-api-in-salesforce/
[5]. https://resources.docs.salesforce.com/latest/latest/en-us/sfdc/pdf/api_rest.pdf
[6]. https://www.integrate.io/blog/salesforce-rest-api-integration/