



PART 1: Designing APIs



Objectives



- Describe REST API architecture
- Describe API development lifecycle
- Translate functional requirements for APIs into resources and HTTP methods
- Navigate Anypoint Platform



Module 1: Introducing RESTful API Design

Goal



SOAP

```

▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://soap.training"
      <xs:element name="findFlight" type="string"/>
      <xs:element name="findFlightResponse" type="string"/>
      <xs:element name="listAllFlights" type="string"/>
      <xs:element name="listAllFlightsResponse" type="string"/>
      ▼<xs:complexType name="listAllFlightsResponse"
        <xs:sequence base="base64Binary" maxOccurs="1" minOccurs="1"/>
      </xs:complexType>
      ▼<xs:complexType name="listAllFlights"
        <xs:sequence base="base64Binary" maxOccurs="1" minOccurs="1"/>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>

```

RPC

```

POST /SendUserMessage HTTP/1.1
Host: api.example.com
Content-type: application/json

{"userId": 501, "message": "Hello!"}

```

REST

```

#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:

```

Objectives



- Describe the common web API formats including SOAP, RPC, and REST
- Describe REST API architecture
- List the rules for retaining REST principles in APIs
- Describe design-first approach for REST APIs

All contents © MuleSoft Inc.

5

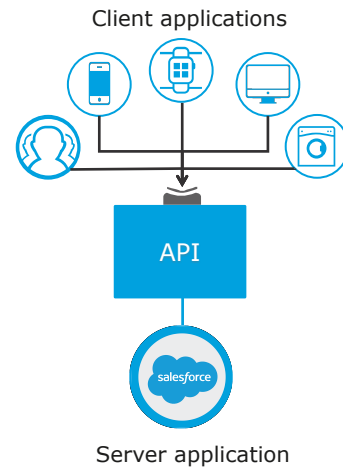
Introducing APIs



Application Programming Interfaces (APIs)



- Interface to allow applications to talk to each other
- It provides information for how to communicate with a software component, defining the
 - Operations (what to call)
 - Inputs (what to send with a call)
 - Outputs (what you get back from a call)
 - Underlying data types



All contents © MuleSoft Inc.

7

Benefits of APIs



- Standard interface for communication
- Easy to use
- Layer of abstraction
 - Acts as a security layer
 - Hides the specifics of the systems talking to each other

All contents © MuleSoft Inc.

8

Terminology used with APIs



- Client
 - Initiating party that sends requests to APIs (many clients can consume an API)
- Server
 - Provides a service by sending a response to the API requests
- Stateless
 - Server does not store client information between multiple requests
- Resources
 - Entities or categories that has a URI to which requests can be sent
- Methods
 - Part of a request that tells the server the action the client wants to perform

All contents © MuleSoft Inc.

9

Types of APIs

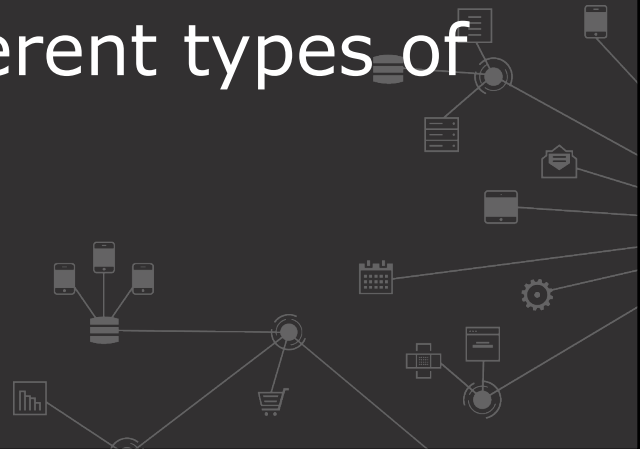


- APIs belong to two categories
 - Traditional server to server, libraries in source code
 - Web APIs (focus will be on this throughout the class)
- Over tens of thousands of public web APIs that help connect applications via the internet exists
- Web APIs are created in a wide range of web formats and standards
 - SOAP (Simple Object Access Protocol) APIs
 - RPC (Remote Procedure Call) APIs
 - JSON-RPC APIs
 - XML-RPC APIs
 - REST (Representational State Transfer) APIs

All contents © MuleSoft Inc.

10

Introducing different types of web APIs



SOAP APIs



- Can be used across different communication protocols (HTTP, SMTP, UDP)
- Require SOAP library in the client to build and receive requests
- Can be stateful or stateless
- Requires extensive programming knowledge
- Rely heavily on XML format for defining a web service (WSDL)
 - XML format results in more lines of code which results in bigger message sizes making them unwieldy

Walkthrough 1-1: Review and make a call to a SOAP API



- Examine an example SOAP API
- Make a call to a SOAP API endpoint to retrieve information

The screenshot shows the WebserviceX.NET website. The main heading is "Explore services for all devices". Below it, there's a description: "The WebserviceX.NET Data Protocol is a SOAP modifying information on the web." and a green "Explore" button. To the right, there's a "Demo" section for the "getAirportInformationByAirportCode" service. It includes a "Test" section with a parameter input field (airportCode: LHR) and an "Invoke" button. Below the "Test" section, there's a "SOAP 1.1" section showing a sample request and response. The response is a SOAP 1.1 envelope with a header and a body containing the airport information.

13

XML/JSON formatted-RPC APIs



- Use HTTP as the communication protocol
- Return data in XML/JSON format
 - Tightly coupled to the RPC (Remote Procedure Call) type
- Require extensive documentation to be used
 - Require developers to know the name of the procedures and specific parameters and the order of the parameters
- Require separate commands for separate actions
- Are stateless
- Are easier for developers to use compared to SOAP APIs

Walkthrough 1-2: Review and make a call to an RPC API



- Examine the Slack RPC API
- Make a call to an RPC endpoint

The screenshot shows the Slack Web API documentation page. On the left is a sidebar with navigation links like 'Start here', 'App features', 'Messages', and 'APIs'. The main content area is titled 'Slack Web API' and includes a 'Basics' section explaining that the API uses HTTP methods and returns JSON. Below this is a code block showing a sample JSON response for the 'channels.list' endpoint. To the right of the documentation is a 'Tester' tab with a form to test the API call. The form has fields for 'token' (Required), 'exclude_archived' (Optional, default=false), and 'Extra args'. A 'Test Method' button is at the bottom right of the form.

Slack Web API

The Slack Web API allows you to build applications that interact with complex ways than the integrations we provide out of the box.

Basics

The Web API consists of HTTP RPC-style methods, all of the form <https://slack.com/api/METHOD>.

All methods must be called using HTTPS. Arguments can be passed as params, or as a mix. The response contains a JSON object, which will always have a boolean property `ok`, indicating success or failure. For failure `ok` property will contain a short machine-readable error code. In the case of a failure that could still be completed successfully, `ok` will be `false`, and `error` property will contain a short machine-readable warning code (or codes, in the case of multiple warnings).

```
{
  "ok": true,
  "channel": "C1234567890"
}
```

```
{
  "ok": false,
  "error": "something_hud"
}
```

```
{
  "ok": true,
  "warning": "something_problematic",
  "stuff": "your requested information"
}
```

channels.list

Documentation Tester

View another method...

Argument	Value
token	Required <input type="text" value="No token"/>
exclude_archived	Optional, default=false <input type="text"/>
Extra args	<input type="text"/>

Generate tokens to test with [here](#).

Test Method

All contents © MuleSoft Inc.

15

REST APIs



- Typically used over HTTP
- Supports any content type (XML, JSON, YAML...)
- Requires single command for multiple actions
- Stateless
- Are less difficult for new developers to use compared to SOAP APIs

****Rest of the modules will focus on REST APIs**

All contents © MuleSoft Inc.

16

Walkthrough 1-3 Review and make a call to a REST API



- Examine the Vimeo REST API
- Make a call to a REST API endpoint to retrieve information

vimeo developer() API Player Guidelines My Apps Help Jobs

Authentication

Getting Started

- Get your API key
- Register a new application
- Get an access token
- Make API calls to Vimeo
- Method
- Endpoint
- Parameters
- Set your API version
- Upload a video
- Stay up to date

Common formats & parameters

Guidelines and best practices

Libraries & SDKs

Embed Videos with oEmbed

Playground

Developer / API

Getting Started

1. Get your API key

You'll need a Client ID and Client Secret with some basic information about your application, you can always edit it.

Register a new application

We also recommend you use on

2. Get an access token

To make API requests you will need token must be authenticated.

Vimeo uses the OAuth 2 specification to choose to accept or deny access. It's a code for your access token.

[See the authentication documentation](#)

API / Playground

categories

(category)

GET https://api.vimeo.com/categories

View all categories

Name	Value	Description	Required
page	int	The page number to show.	No
per_page	int	Number of items to show on each page. Max 100.	No
sort	Choose one	Technique used to sort the results.	No
direction	string	The direction that the results are sorted.	No

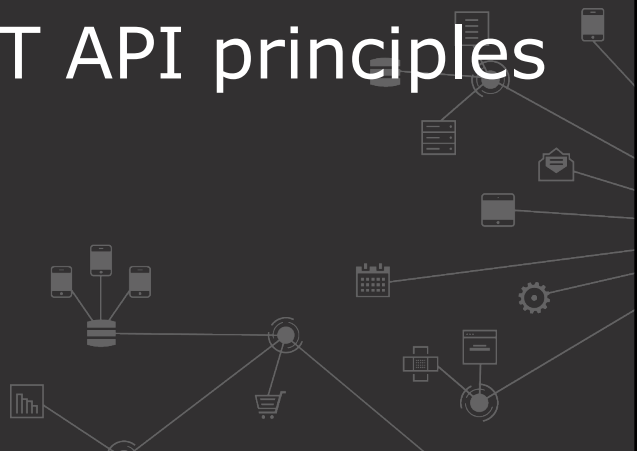
Use Application: API Playground

Make Call

All contents © MuleSoft Inc.

17

Introducing REST API principles



Enforcing REST principles in an API



- Separation of concerns between client and server
 - Changes to the client application should not affect the server application and vice versa
 - The applications must grow and scale independently
- Uniform interface
 - Allows client and server to communicate using a single language, independent of the architecture of each of them
 - Provides standard means of communication by decoupling implementation
- Stateless
 - The API should not rely on server to store the client session data to make a call
 - The API should provide data in that call itself
 - To reduce memory, session information is stored in the client

All contents © MuleSoft Inc.

19

Enforcing REST principles in an API



- Client-side caching
 - The API should encourage clients to cache data on their side
 - This is a result of increased request overhead due to the REST principle of statelessness
 - Helps reduce number of interactions with the API
 - Provides API consumers with tools to deliver fast and efficient applications
- Layered system
 - Creates scalable, modular applications promoting loose coupling
 - Promotes flexibility, and security through multiple layers
 - Reduce cost, increase delivery speed and maintainability

All contents © MuleSoft Inc.

20

Enforcing REST principles in an API (cont)



- Code on demand
 - Allows for code to be sent via the API for using within an application
 - For example, a client application can send a request to a server to receive code that can be executed locally to utilize the resources that the client application has access to
 - Increases time efficiency to add new features, extensibility and configurability
 - Optional principle to implement in APIs (since it reduces visibility)

Planning to design REST APIs



Introducing API-first design approach



- Consumer driven
 - Start by figuring out what API consumers really want from your API
 - Think about design that best serves them
 - Define the interface between infrastructure and the API consumers first
- Focus exclusively on design
 - API design should not be constrained by existing limitations in the systems or by legacy infrastructure
 - These constraints should come into the picture when time comes to implement the API
 - Model cleanly and consistently
 - Design iteratively and get feedback from API consumers on its usability and functionality along the way
 - Include tools to discover and learn the API

All contents © MuleSoft Inc.

API consumer focused design considerations



- Audience
 - Who is the API for?
- Actions
 - Which actions do they need access to?
- Interactions
 - Explain how the API will interact with existing backend services or applications
- Maintenance
 - How are you going to maintain the API?
- Versioning
 - How are you going to version the API?

All contents © MuleSoft Inc.

API consumer focused design considerations



- Documentation
 - How are you going to document the API?
- Access
 - How will API consumers access/interact with the API?
- Support
 - How are you going to manage support?

Summary



Summary



- Three common web API formats
 - SOAP
 - RPC
 - REST
- REST API architecture allows for various constraints and governance
- Highly recommended to take an API-first design approach with the following considerations in mind
 - Take an approach of exposing data in the system through APIs
 - Design for the API consumer
 - Prototype designs and work with key stakeholders before implementation
 - Design for maintainability and long-term usability