What is a Composite API?

Composite APIs are a design approach to batch API requests sequentially into a single API call. Rather than multiple round trips to a server, a client can make one API request with a chain of calls and receive one response.

For example, to create an order within a shopping cart API, you may require several endpoints:

- Create customer
- Create order for customer
- Add item to order
- Add another item
- Change order status

While it is useful to have granular access to call a subset of the above sequence, together they comprise a common shopping cart use case. In order to fulfill an order, you'll need to make these calls in succession.

A composite API provides a design solution that is more efficient for the client. Instead of five or more requests, there is only one. Rather than parsing five or more responses, all the necessary data is passed at the end of the sequence.

Example Composite API

Building on the shopping cart example, below you can see an example request and response for a two-step composite API call. There are many considerations for composite API design, which will be covered in the next section. This example shows one approach to a composite API that creates a new customer, then references that customer in a new order.

```
POST /composite
{
  "composite-request": [
    {
      "method": "POST",
      "path": "/customer",
      "ref": "newcustomer",
      "body": {"name": "Tony Stark"}
    },
    {
      "method": "POST",
      "path": "/order",
      "ref": "neworder",
      "body": {"customer": "@{newcustomer.id}"}
    }
  ]
}
```

In this example, the /composite endpoint receives a POST request, which includes an object in the body. Within the object is a single field that includes an array of individual requests. Each request provides a method, path, ref, and body. The ref is used to reference data provided in earlier steps, as well as within the composite API response.

Here's how a successful response might look from this example call:

```
{
  "success": true,
  "composite-response": [
    {
      "ref": "newcustomer",
      "body": {"id": 123456}
      "success": true,
      "status": 201
    },
    {
      "ref": "neworder",
      "body": {"id": 234567}
      "success": true,
      "status": 201
    }
  ]
}
```

The structure of the composite response echos the request. There's a top-level success field followed by an object that contains an array of individual responses. Each step includes ref, body, success, and status fields.

This example makes a lot of assumptions about how you might design your own composite APIs. The next section covers some of the elements to consider.

How to Design a Composite API

As you've seen in the example, a composite API does not require you to create a brand new API. It will most likely augment an existing REST API design. In fact, you may have an API in production with many consumers before you recognize the need for composite features. Multiple calls in rapid sequence is a sure sign you should consider including a composite API into

your next design.

Most likely, your composite API will be a single endpoint to which you'll POST your sequence of requests. Even with these criteria, you have some considerations for your composite API design:

- *Request representation*: At a minimum, you need an array of objects that include the endpoint, method, and a way to pass data or parameters with each request.
- *Authentications*: Will credentials be passed from the composite call to individual requests, or do you also need granular authentication for each underlying call?
- *Response rendering*: You need to at least include the response from the final request. Likely, you also need to include some subset of response data from previous steps.
- *Field references*: Sequential calls will be most useful if they can include data from earlier steps, which means you'll need to determine how to structure those references.
- *Error handling*: If one or more calls in your sequence returns an error, you'll need to provide details to help consumers debug the response.
- *Partial successes*: In addition to reporting errors, you'll need to decide whether to continue the steps after the error.
- *All or none*: Some use cases might also expect transactional features, where an error "rolls back" any changes in previous steps.

The decisions you make based on the above considerations will impact the design of your composite API. How you support composite calls will be determined by the use cases you expect.

Batch APIs vs Composite APIs

Depending on your needs, a batch API may meet the use cases you would otherwise point to a composite API. At first glance, they may seem similar. In fact, a composite API fulfills the requirements of a batch API, but the opposite is not true.

Batch APIs group API calls that aren't necessarily related. For example, rather than making 10 API calls to create 10 new customers, a batch API endpoint would achieve the same result in a single call.

Some key differences in batch APIs make them easier to implement:

- A subset of calls can fail without impacting the others
- Subsequent calls do not need to reference previous data
- If a call fails, there is no need to roll back changes made by earlier steps

Composite APIs provide a larger number of use cases, so they will likely be more powerful for consumers. However, batch APIs are an excellent option if the sequence and reference features aren't needed to meet your common use cases.