**How to design a REST API**

Learning REST in pieces is one thing, while applying all those learned concepts into real application design is completely another challenge. In this tutorial, we will **learn to design REST APIs** for a network-based application. Please note that the takeaway from this whole exercise is the learning of **how to apply REST principles in design process**.

**Steps in designing REST Services**

[Identify Object Model](https://restfulapi.net/rest-api-design-tutorial-with-example/#object-model)

[Create Model URIs](https://restfulapi.net/rest-api-design-tutorial-with-example/#model-uris)

[Determine Representations](https://restfulapi.net/rest-api-design-tutorial-with-example/#create-representations)

[Assign HTTP Methods](https://restfulapi.net/rest-api-design-tutorial-with-example/#assign-http-methods)

[More Actions](https://restfulapi.net/rest-api-design-tutorial-with-example/#more-actions)

**Identify Object Model**

The very first step in designing a REST API based application is – identifying the objects which will be presented as **resources**.

For **a network-based application**, object modeling is pretty much more straightforward. There can be many things such as devices, managed entities, routers, modems, etc. For simplicity sake, we will consider only two resources i.e.

* Devices
* Configurations

Here configuration is sub-resource of a device. A device can have many configuration options.

Note that both objects/resources in our above model will have a unique identifier, which is the integer id property.

**Create Model URIs**

Now when the object model is ready, it’s time to decide the resource URIs. At this step, while [designing the resource URIs](https://restfulapi.net/resource-naming/) – focus on the relationship between resources and its sub-resources. These **resource URIs are endpoints for RESTful services**.

In our application, a device is a top-level resource. And configuration is sub-resource under the device. Let’s write down the URIs.

|  |
| --- |
| /devices  /devices/{id}    /configurations  /configurations/{id}    /devices/{id}/configurations  /devices/{id}/configurations/{id} |

Notice that these **URIs do not use any verb or operation**. It’s crucial not to include any verb in URIs. URIs should all be nouns only.

**Determine Representations**

Now when resource URIs have been decided, let’s work on their representations. Mostly representations are defined in either [XML or JSON](https://restfulapi.net/content-negotiation/) format. We will see XML examples as its more expressive on how data is composed.

**Collection of Device Resource**

When returning a collection resource, include only the most important information about resources. This will keep the size of payload small, and so will improve the **performance of REST APIs**.

|  |
| --- |
| **<devices size="2">**        <link rel="self" href="/devices"/>    **<device id="12345">**          <link rel="self" href="/devices/12345"/>          <deviceFamily>apple-es</deviceFamily>          <OSVersion>10.3R2.11</OSVersion>          <platform>SRX100B</platform>          <serialNumber>32423457</serialNumber>          <connectionStatus>up</connectionStatus>          <ipAddr>192.168.21.9</ipAddr>          <name>apple-srx\_200</name>          <status>active</status>      </device>    **<device id="556677">**          <link rel="self" href="/devices/556677"/>          <deviceFamily>apple-es</deviceFamily>          <OSVersion>10.3R2.11</OSVersion>          <platform>SRX100B</platform>          <serialNumber>6453534</serialNumber>          <connectionStatus>up</connectionStatus>          <ipAddr>192.168.20.23</ipAddr>          <name>apple-srx\_200</name>          <status>active</status>      </device>    **</devices>** |

**Single Device Resource**

Opposite to collection URI, here include complete information of a device in this URI. Here, also include a list of links for sub-resources and other supported operations. This will make your REST API [HATEOAS](https://restfulapi.net/hateoas/) driven.

# **Configuration under device**

|  |
| --- |
| **<device id="12345">**  **<link rel="self" href="/devices/12345"/>**    **<id>12345</id>**  **<deviceFamily>apple-es</deviceFamily>**  **<OSVersion>10.0R2.10</OSVersion>**  **<platform>SRX100-LM</platform>**  **<serialNumber>32423457</serialNumber>**  **<name>apple-srx\_100\_lehar</name>**  **<hostName>apple-srx\_100\_lehar</hostName>**  **<ipAddr>192.168.21.9</ipAddr>**  **<status>active</status>**    **<configurations size="2">**  **<link rel="self" href="/configurations" />**    **<configuration id="42342">**  **<link rel="self" href="/configurations/42342" />**  **</configuration>**    **<configuration id="675675">**  **<link rel="self" href="/configurations/675675" />**  **</configuration>**  **</configurations>**    **<method href="/devices/12345/exec-rpc" rel="rpc"/>**  **<method href="/devices/12345/synch-config"rel="synch device configuration"/>**  **</device>** |

**Configuration Resource Collection**

Similar to device collection representation, create configuration collection representation with only minimal information.

**# Configurations (Standalone)**

|  |
| --- |
| **<configurations size="20">**  **<link rel="self" href="/configurations" />**    **<configuration id="42342">**  **<link rel="self" href="/configurations/42342" />**  **</configuration>**    **<configuration id="675675">**  **<link rel="self" href="/configurations/675675" />**  **</configuration>**  **...**  **...**  **</configurations>** |

Please note that configurations collection representation inside device is similar to top-level configurations URI. Only difference is that configurations for a device are only two, so only two configuration items are listed as subresource under device.

**Single Configuration Resource**

Now, single configuration resource representation must have all possible information about this resource – including relevant links.

|  |
| --- |
| **<configuration id="42342">**  **<link rel="self" href="/configurations/42342" />**  **<content><![CDATA[...]]></content>**  **<status>active</status>**  **<link  rel="raw configuration content" href="/configurations/42342/raw" />**  **</configuration>** |

**Configuration Resource Collection Under Single Device**

This resource collection of configurations will be a subset of the primary collection of configurations, and will be specific a device only. As it is the subset of primary collection, **DO NOT create a different representation data fields** than primary collection. Use the same presentation fields as the primary collection.

|  |
| --- |
| <configurations size="2">      <link rel="self" href="/devices/12345/configurations" />        <configuration id="53324">          <link rel="self" href="/devices/12345/configurations/53324" />          <link rel="detail" href="/configurations/53324" />      </configuration>        <configuration id="333443">          <link rel="self" href="/devices/12345/configurations/333443" />          <link rel="detail" href="/configurations/333443" />      </configuration>  </configurations> |

Notice that this subresource collection has two links. One for its direct representation inside sub-collection i.e. /devices/12345/configurations/333443 and other pointing to its location in **primary collection** i.e. /configurations/333443.

Having two links is essential as you can provide access to a device- specific configuration in a more unique manner, and you will have the ability to mask some fields (if design requires it), which shall not be visible in a **secondary collection**.

**Single Configuration Resource Under Single Device**

This representation should have either exactly similar representation as of Configuration representation from the primary collection, OR you may mask few fields.

This subresource representation will also have an additional link to its primary presentation.

|  |
| --- |
| <configuration id="11223344">      <link rel="self" href="/devices/12345/configurations/11223344" />      <link rel="detail" href="/configurations/11223344" />      <content><![CDATA[...]]></content>      <status>active</status>      <link rel="raw configuration content" href="/configurations/11223344/raw" />  </configuration> |

Now, before moving forward to the next section, let’s note down a few observations, so you don’t miss them.

* **Resource URIs are all nouns.**
* **URIs are usually in two forms – collection of resources and singular resource.**
* **Collection may be in two forms primary collection and secondary collection. Secondary collection is sub-collection from a primary collection only.**
* **Each resource/collection contain at least one link i.e. to itself.**
* **Collections contain only most important information about resources.**
* **To get complete information about a resource, you need to access through its specific resource URI only.**
* **Representations can have extra links (i.e. methods in single device). Here method represent a POST method. You can have more attributes or form links in altogether new way also.**
* **We have not talked about operations on these resources yet.**

**Assign HTTP Methods**

So our resource URIs and their representation are fixed now. Let’s decide the possible operations in the application and map these operations on resource URIs. A user of our network application can perform browse, create, update, or delete operations. So let’s assign them.

**Browse all devices or configurations [Primary Collection]**

|  |
| --- |
| HTTP GET /devices  HTTP GET /configurations |

If the collection size is large, you can apply **paging and filtering** as well. e.g., Below requests will fetch the first 20 records from collection.

|  |
| --- |
| HTTP GET /devices**?startIndex=0&size=20**  HTTP GET /configurations**?**startIndex=0**&**size=20 |

**Browse all devices or configurations [Secondary Collection]**

|  |
| --- |
| HTTP GET /devices/{id}/configurations |

It will be mostly a small size collection – so no need to enable filtering or sorting here.

**Browse single device or configuration [Primary Collection]**

To get the complete detail of a device or configuration, use GET operation on singular resource URIs.

|  |
| --- |
| HTTP GET /devices/{id}  HTTP GET /configurations/{id} |

**Browse single device or configuration [Secondary Collection]**

|  |
| --- |
| HTTP GET /devices/{id}/configurations/{id} |

Sub-resource representation will be either same as or a subset of primary presentation.

**Create a device or configuration**

Create is **NOT** [idempotent](https://restfulapi.net/idempotent-rest-apis/) operation, and in HTTP protocol – POST is also not idempotent. So use POST.

|  |
| --- |
| HTTP POST /devices  HTTP POST /configurations |

Please note that request payload will not contain any id attribute, as the server is responsible for deciding it.

**The response to create request (HTTP POST /configurations)** will look like this:

It include

* 201 Created
* Location header for new resource: Location: [http://example.com/network-app/configurations/**678678**](http://example.com/network-app/configurations/678678)
* Complete representation of new resource.

|  |
| --- |
| HTTP/1.1 201 Created  Content-Type: application/xml  Location: [http://example.com/network-app/configurations/**678678**](http://example.com/network-app/configurations/678678)    **<configuration id="678678">**  **<link rel="self" href="/configurations/678678" />**  **<content><![CDATA[...]]></content>**  **<status>active</status>**  **<link  rel="raw configuration content" href="/configurations/678678/raw" />**  **</configuration>** Updates & creation should return a resource representation A PUT, POST or PATCH call may make modifications to fields of the underlying resource that weren't part  of the provided parameters (for example: created\_at or updated\_at timestamps).  To prevent an API consumer from having to hit the API again for an updated representation, have the API  **return the updated (or created) representation as part of the response**.  In case of a POST that resulted in a creation, use a [HTTP 201 status code](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5) and include a [**Location header**](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.30)  that points to the URL of the new resource. Both of those in should be in addition to including **the newly**  **created resource representation** as the body of the response. Limiting which fields are returned by the API The API consumer doesn't always need the full representation of a resource. The ability to select and  chose returned fields goes a long way in letting the API consumer minimize network traffic  and speed up their own usage of the API.  Use a fields query parameter that takes a comma separated list of fields to include. For example,  the following request would retrieve just enough information to display a sorted listing of open tickets:  GET /tickets**?**fields=id,subject,updated\_at**&**state=open**&**sort=-updated\_at  **Note**: This approach can also be combined with with [autoloading of related resources](https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#autoloading):  GET /tickets**?**embed=customer**&**fields=id,customer.id,customer.name |

**Update a device or configuration**

Update operation is an idempotent operation and HTTP PUT is also is idempotent method. So we can use PUT method for update operations.

|  |
| --- |
| HTTP PUT /devices/{id}  HTTP PUT /configurations/{id} |

**PUT response** may look like this.

* 200 OK
* Complete representation of new resource.

|  |
| --- |
| HTTP/1.1 200 OK  Content-Type: application/xml    **<configuration id="678678">**  **<link rel="self" href="/configurations/678678" />**  **<content><![CDATA[. updated content here .]]></content>**  **<status>active</status>**  **<link  rel="raw configuration content" href="/configurations/678678/raw" />**  **</configuration>** |

**Remove a device or configuration**

Removing is always a DELETE operation.

|  |
| --- |
| HTTP DELETE /devices/{id}  HTTP DELETE /configurations/{id} |

A successful response SHOULD be **202 (Accepted) if resource has been queues for deletion (async operation),**

or **200 (OK) / 204 (No Content) if resource has been deleted permanently (sync operation).**

In the case of async operation, the application shall return a task id that can be tracked for success/failure status.

*Please note that you should put enough analysis in deciding the behavior when a subresource is deleted from the system. Usually, you may want to****SOFT DELETE****a resource in these requests – in other words, set their status INACTIVE. By following this approach, you will not need to find and remove its references from other places as well.*

**Applying or Removing a configuration from a device**

In a real application, you will need to apply the configuration on the device – OR you may want to remove the configuration from the device (not from the primary collection). You shall use PUT and DELETE methods in this case, because of its **idempotent** nature.

|  |
| --- |
| //Apply Configuration on a device  HTTP PUT /devices/{id}/configurations    //Remove Configuration on a device  HTTP DELETE /devices/{id}/configurations/{id} |