# Asynchronous Processing

#### Problem

RESTful APIs use HTTP, which is a synchronous protocol.  An application sends requests, blocks, and waits for a response. However,  in some cases when the process has the potential to take a significantly long time (due to a slow backend system), this is not the desired behavior. Typical examples are mobile or modern Web application that provides reactive UI. Such an application can submit a request to create a new object, continue with some other task, and only show a notification when the customer object is created in the backend system and ready to consume.

#### Solution

Design the API so that an application can trigger asynchronous work. And only track the processing status from time to time. When the result is ready, the application should be able to retrieve it, or if the processing is not required any more, the application should be able to cancel it.

As in the previous case, this can be done using standard HTTP headers and status codes. For that reason, we use HTTP code 202 Accepted to inform the application that its request has been accepted for processing.

The W3C’s HTTP standard (<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>) defines 202 code as follows:

**202 Accepted**

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response SHOULD include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

The W3C’s HTTP standard (<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>) defines 303 code as follows: The Location header is used to provide a link to such a status monitor. The application can use the URI provided in the Location header to poll the status of the processing. Once the processing is finished, the API provides the location of the actual result again using Location header but with a different HTTP status code, namely 303.

**303 See Other**

The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource.

The 303 response MUST NOT be cached but the response to the second request might be cacheable.  
The different URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI.

Note: Many pre-HTTP/1.1 user agents do not understand the 303 status. When interoperability with such clients is a concern the 302 status code may be used instead since most user agents react to a 302 response as described here for 303.

DELETE /resources/123/status request: In some cases, the application may want to cancel the processing. This could be achieved by sending

The following RAML snippet demonstrates the pattern:

#%RAML 1.0

title: Example Asynchronous API

version: v1

mediaType: application/json

types:

Data: object

Status:

properties:

status:

enum: [working, canceled, failed, success]

/resources:

post:

body:

type: Data

responses:

202:

description: |

The request has been accepted for processing. Use the URI provided in the Location: header to monitor the status of the processing

headers:

Location:

example: /resources/123/status

/{id}:

get:

responses:

200:

body: Data

/status:

get:

responses:

200:

description: |

The API is still processing the request.

body:

type: Status

example: {"status": "working"}

303:

headers:

Location:

example: /resources/123

description: |

The processing has finished successfully.Use the URI provided in the Location: to obtain the actual result

body:

type: Status

example: {"status": "success"}

500:

description: |

The processing has failed

body:

type: Status

example: {"status": "failed"}

delete:

responses:

200:

description: |

The processing has been canceled

body:

type: Status

example: {"status": "canceled"}