Network Working Group J. Jena
Internet-Draft
Intended status: Standards Track S. Dalal
Expires: 13 May 2023 9 November 2022

The Idempotency-Key HTTP Header Field
draft-ietf-httpapi-idempotency-key-header-02

Abstract

The HTTP Idempotency-Key request header field can be used to carry idempotency key in order to make non-idempotent HTTP methods such as "POST" or "PATCH" fault-tolerant.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Building Blocks for HTTP APIs Working Group mailing list (httpapi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/httpapi/.

Source for this draft and an issue tracker can be found at https://github.com/mnot/idempotency.

Status of This Memo

This Internet-Draft will expire on 13 May 2023.

Copyright Notice

Table of Contents

1. Introduction

Idempotence is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application. It does not matter if the operation is called only once, or 10s of times over. The result SHOULD be the same.

Idempotency is important in building a fault-tolerant HTTP API. An HTTP request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. According to [RFC7231], HTTP methods "OPTIONS", "HEAD", "GET", "PUT" and "DELETE" are idempotent while methods "POST" and "PATCH" are not.

Let's say a client of an HTTP API wants to create (or update) a resource using a "POST" method. Since "POST" is NOT an idempotent method, calling it multiple times can result in duplication or wrong updates. Consider a scenario where the client sent a "POST" request to the server, but it got a timeout. Following questions arise : Is the resource actually created (or updated)? Did the timeout occur during sending of the request, or when receiving of the response? Can the client safely retry the request, or does it need to figure out what happened in the first place? If "POST" had been an idempotent method, such questions may not arise. Client would safely retry a request until it actually gets a valid response from the server.

For many use cases of HTTP APIs, duplicated resources are a severe problem from a business perspective. For example, duplicate records for requests involving any kind of money transfer "MUST NOT" be allowed. In other cases, processing of duplicate webhook delivery is not expected.

1.1. Notational Conventions

{::boilerplate bcp14-tagged}

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and includes, by reference, the IMF-fixdate rule as defined in Section 7.1.1.1 of [RFC7231].

The term "resource" is to be interpreted as defined in Section 2 of [RFC7231], that is identified by an URI. The term "resource server" is to be interpreted as "origin server" as defined in Section 3 of [RFC7231].

2. The Idempotency-Key HTTP Request Header Field

An idempotency key is a unique value generated by the client which the resource server uses to recognize subsequent retries of the same request. The "Idempotency-Key" HTTP request header field carries this key.

2.1. Syntax

"Idempotency-Key" is an Item Structured Header [RFC8941]. Its value MUST be a String. Refer to Section 3.3.3 of [RFC8941] for ABNF of "sf-string":

```
Idempotency-Key = sf-string
```

Clients MUST NOT include more than one "Idempotency-Key" header field in the same request.

The following example shows an idempotency key using "UUID" [RFC4122]:

```
Idempotency-Key: "8e03978e-40d5-43e8-bc93-6894a57f9324"
```

2.2. Uniqueness of Idempotency Key

The idempotency key that is supplied as part of every "POST" request MUST be unique and MUST NOT be reused with another request with a different request payload.

Uniqueness of the key MUST be defined by the resource owner and MUST be implemented by the clients of the resource server. It is RECOMMENDED that "UUID" [RFC4122] or a similar random identifier be used as an idempotency key.

2.3. Idempotency Key Validity and Expiry

The resource MAY enforce time based idempotency keys, thus, be able to purge or delete a key upon its expiry. The resource server SHOULD define such expiration policy and publish it in the documentation.

2.4. Idempotency Fingerprint

An idempotency fingerprint MAY be used in conjunction with an idempotency key to determine the uniqueness of a request. Such a fingerprint is generated from request payload data by the resource server. An idempotency fingerprint generation algorithm MAY use one of the following or similar approaches to create a fingerprint.

- Checksum of the entire request payload.

- Checksum of selected element(s) in the request payload.

- Field value match for each field in the request payload.

- Field value match for selected element(s) in the request payload.

- Request digest/signature.

2.5. Responsibilities

Client

Clients of HTTP API requiring idempotency, SHOULD understand the idempotency related requirements as published by the server and use appropriate algorithm to generate idempotency keys.

For each request, client SHOULD

- Send a unique idempotency key in the HTTP "Idempotency-Key" request header field.

Resource Server

Resource server MUST publish idempotency related specification. This specification MUST include expiration related policy if applicable. Server is responsible for managing the lifecycle of the idempotency key.

For each request, server SHOULD

- Identify idempotency key from the HTTP "Idempotency-Key" request header field.

- Generate idempotency fingerprint if required.

- Check for idempotency considering various scenarios including the ones described in section below.

2.6. Idempotency Enforcement Scenarios

- First time request (idempotency key and fingerprint has not been seen)

  The resource server SHOULD process the request normally and respond with an appropriate response and status code.

- Duplicate request (idempotency key and fingerprint has been seen)

  Retry

  The request was retried after the original request completed. The resource server SHOULD respond with the result of the previously completed operation, success or an error. See Error Scenarios for details on errors.

  Concurrent Request

  The request was retried before the original request completed. The resource server MUST respond with a resource conflict error. See Error Scenarios for details.

2.7. Error Scenarios

If the "Idempotency-Key" request header is missing for a documented idempotent operation requiring this header, the resource server SHOULD reply with an HTTP "400" status code with body containing a link pointing to relevant documentation. Following examples shows an error response describing the problem using [RFC7807].

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json
Content-Language: en
{
  "type": "https://developer.example.com/idempotency",
  "title": "Idempotency-Key is missing",
  "detail": "This operation is idempotent and it requires correct usage of Idem
}
```

Alternately, using the HTTP header "Link", the client can be informed about the error as shown below.

```
HTTP/1.1 400 Bad Request
Link: <https://developer.example.com/idempotency>;
  rel="describedby"; type="text/html"
```

If there is an attempt to reuse an idempotency key with a different request payload, the resource server SHOULD reply with a HTTP "422" status code with body containing a link pointing to relevant documentation. The status code "422" is defined in Section 11.2 of [RFC4918].

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en
{
  "type": "https://developer.example.com/idempotency",
  "title": "Idempotency-Key is already used",
  "detail": "This operation is idempotent and it requires correct usage of Idem
}
```

The server can also inform the client by using the HTTP header "Link" as shown below.

```
HTTP/1.1 422 Unprocessable Entity
Link: <https://developer.example.com/idempotency>;
rel="describedby"; type="text/html"
```

If the request is retried, while the original request is still being processed, the resource server SHOULD reply with an HTTP "409" status code with body containing problem description.

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json
Content-Language: en
{
  "type": "https://developer.example.com/idempotency",
  "title": "A request is outstanding for this Idempotency-Key",
  "detail": "A request with the same Idempotency-Key for the same operation is
}
```

Or, alternately using the HTTP header "Link" pointing to the relevant documentation

```
HTTP/1.1 409 Conflict
Link: <https://developer.example.com/idempotency>;
rel="describedby"; type="text/html"
```

Error scenarios above describe the status of failed idempotent requests after the resource server prcocesses them. Clients MUST correct the requests (with the execption of 409 where no correction is required) before performing a retry operation, or the the resource server MUST fail the request and return one of the above errors.

For other 4xx/5xx errors, such as 401, 403, 500, 502, 503, 504, 429, or any other HTTP error code that is not listed here, the client SHOULD act appropriately by following the resource server's documentation.

3. IANA Considerations

3.1. The Idempotency-Key HTTP Request Header Field

The Idempotency-Key field name should be added to the "Hypertext Transfer Protocol (HTTP) Field Name Registry".

Field Name: Idempotency-Key

Status: permanent

Specification document: This specification, Section 2

4. Implementation Status

Note to RFC Editor: Please remove this section before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Organization: Stripe

- Description: Stripe uses custom HTTP header named "Idempotency-Key"

- Reference: https://stripe.com/docs/idempotency

Organization: Adyen

- Description: Adyen uses custom HTTP header named "Idempotency-Key"

- Reference: https://docs.adyen.com/development-resources/api-idempotency/

Organization: Dwolla

- Description: Dwolla uses custom HTTP header named "Idempotency-Key"

- Reference: https://docs.dwolla.com/

Organization: Interledger

- Description: Interledger uses custom HTTP header named "Idempotency-Key"

- Reference: https://github.com/interledger/

Organization: WorldPay

- Description: WorldPay uses custom HTTP header named "Idempotency-Key"

- Reference: https://developer.worldpay.com/docs/wpg/idempotency

Organization: Yandex

- Description: Yandex uses custom HTTP header named "Idempotency-Key"

- Reference: https://cloud.yandex.com/docs/api-design-guide/concepts/idempotency

Organization: http4s.org

- Description: Http4s is a minimal, idiomatic Scala interface for HTTP services.

- Reference: https://github.com/http4s/http4s

Organization: Finastra

- Description: Finastra uses custom HTTP header named "Idempotency-

Key"

- Reference: https://developer.fusionfabric.cloud/

Organization: Datatrans

- Description: Datatrans focuses on the technical processing of payments, including hosting smart payment forms and correctly routing payment information.

- Reference: https://docs.datatrans.ch/docs/api-endpoints

4.1. Implementing the Concept

This is a list of implementations that implement the general concept, but do so using different mechanisms:

Organization: Django

- Description: Django uses custom HTTP header named "HTTP_IDEMPOTENCY_KEY"

- Reference: https://pypi.org/project/django-idempotency-key

Organization: Twilio

- Description: Twilio uses custom HTTP header named "I-Twilio-Idempotency-Token" in webhooks

- Reference: https://www.twilio.com/docs/usage/webhooks/webhooks-connection-overrides

Organization: PayPal

- Description: PayPal uses custom HTTP header named "PayPal-Request-Id"

- Reference: https://developer.paypal.com/docs/business/develop/idempotency

Organization: RazorPay

- Description: RazorPay uses custom HTTP header named "X-Payout-Idempotency"

- Reference: https://razorpay.com/docs/razorpayx/api/idempotency/

Organization: OpenBanking

- Description: OpenBanking uses custom HTTP header called "x-idempotency-key"

- Reference: https://openbankinguk.github.io/read-write-api-site3/v3.1.6/profiles/read-write-data-api-profile.html#request-headers

Organization: Square

- Description: To make an idempotent API call, Square recommends adding a property named "idempotency_key" with a unique value in the request body.

- Reference: https://developer.squareup.com/docs/build-basics/using-rest-api

Organization: Google Standard Payments

- Description: Google Standard Payments API uses a property named "requestId" in request body in order to provider idempotency in various use cases.

- Reference: https://developers.google.com/standard-payments/payment-processor-service-api/rest/v1/TopLevel/capture

Organization: BBVA

- Description: BBVA Open Platform uses custom HTTP header called "X-Unique-Transaction-ID"

- Reference: https://bbvaopenplatform.com/apiReference/APIbasics/content/x-unique-transaction-id

Organization: WebEngage

- Description: WebEngage uses custom HTTP header called "x-request-id" to identify webhook POST requests uniquely to achieve events idempotency.

- Reference: https://docs.webengage.com/docs/webhooks

5. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns specific to the idempotency keys.

Resource servers that do not implement strong idempotency keys, such as UUIDs, or have appropriate controls to validate the idempotency keys, could be victim to various forms of security attacks from malicious clients:

- Injection attacks-When the resource server does not validate the idempotency key in the client request and performs a idempotent cache lookup, there can be security attacks (primarily in the form of injection), compromising the server.

- Data leaks-When an idempotency implementation allows low entropy keys, attackers MAY determine other keys and use them to fetch existing idempotent cache entries, belonging to other clients.

To prevent such situations, the specification recommends the following best practices for idempotency key implementation in the resource server.

- Establish a fixed format for the idempotency key and publish the key's specification.

- Always validate the key as per its published specification before processing any request.

- On the resource server, implement a unique composite key as the idempotent cache lookup key. For example, a composite key MAY be implemented by combining the idempotency key sent by the client with other client specific attributes known only to the resource server.

6. Examples

The first example shows an idempotency-key header field with key value using UUID version 4 scheme:

```
Idempotency-Key: "8e03978e-40d5-43e8-bc93-6894a57f9324"
```

Second example shows an idempotency-key header field with key value using some random string generator:

```
Idempotency-Key: "clkyoesmbgybucifusbbtdsbohtyuuwz"
```

7. References

7.1. Normative References

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://datatracker.ietf.org/doc/html/rfc4122>.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <https://datatracker.ietf.org/doc/html/rfc4918>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://datatracker.ietf.org/doc/html/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://datatracker.ietf.org/doc/html/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://datatracker.ietf.org/doc/html/rfc7231>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://datatracker.ietf.org/doc/html/rfc7807>.

[RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://datatracker.ietf.org/doc/html/rfc8941>.

7.2. Informative References

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://datatracker.ietf.org/doc/html/rfc7942>.

Appendix A. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII character).

The rules below are defined in [RFC7230]:

```
 obs-text      = <obs-text, see [RFC7230], Section 3.2.6>
```

Acknowledgments

The authors would like to thank Mark Nottingham for his support for this Internet Draft. We would like to acknowledge that this draft is inspired by Idempotency related patterns described in API documentation of PayPal (https://github.com/paypal/api-standards/blob/master/patterns.md#idempotency) and Stripe (https://stripe.com/docs/idempotency) as well as Internet Draft on

POST Once Exactly (https://tools.ietf.org/html/draft-nottingham-http-poe-00) authored by Mark Nottingham.

The authors take all responsibility for errors and omissions.

Authors' Addresses

Jayadeba Jena

Email: jayadebaj@gmail.com

Sanjay Dalal

Email: sanjay.dalal@cal.berkeley.edu
URI: https://github.com/sdatspun2