



# Origins

- *Representational State Transfer* – REST: defined in 2000 Roy Fielding's PhD dissertation (after he worked on HTTP 1.1 and URI RFCs)
- Web application =
  - network of Web resources (a virtual state-machine)
  - where the user progresses through the application by selecting resource identifiers and resource operations (application state transitions), resulting in the next resource's representation (the next application state) being transferred to the end user for their use.
- An architectural style, not a standard nor a protocol

# Principles of RESTful Architecture (1/2)

- A resource
  - is identified using an URI,
  - references
    - one entity (eg. user Paul) or
    - a set of entities (eg. all male users)
  - URI doesn't change (but the referenced entity might)
  - and can have multiple representations (JSON, XML...).
- The representation of a resource contains enough information for the client to request a change to its state.
  - Messages include enough information to describe how to process them (eg. Content type)
  - HATEOS (*Hypermedia as the Engine of Application State*)

# HATEOAS Example

request

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

response if balance > 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
  <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
  <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
  <link rel="close" href="https://bank.example.com/accounts/12345/status" />
</account>
```

response if balance < 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
</account>
```

# Principles of RESTful Architecture (2/2)

- Separation of concerns between the client (user interface concerns) and the server (data storage and processing concerns)
- Stateless communication: the server only stores resources states while the client is in charge of providing the application state.
- Responses should define the extent to which they can be cached.
- A client may not be directly connected to the end-server: there can be proxies, an additional security layer, and the server might call other servers to complete the service.

# Semantics of HTTP methods

HTTP method	Operation on the resource	URIs: examples	HTTP response status	<i>location</i> header	safe	idem potent
GET	read	GET /serv/users GET /serv/users/34	200	no	yes	Yes
POST	create	POST /serv/users # body { name: "Toto" }	201	Yes	no	no
PUT	update	PUT /serv/users/34 # body { name: "Jacques" }	200, 204	no	no	Yes
DELETE	delete	DELETE /serv/users/34	200, 204, 202	no	no	yes

# Example of scenario

- Book a room:

POST [http://myhotel.com/reservations?date="12/03/2021"&nights=2&persons=4](http://myhotel.com/reservations?date=12/03/2021&nights=2&persons=4)

Server replies with reservation number 123

- Display reservation:

GET <http://myhotel.com/reservations/123>

- Update the reservation:

PUT <http://myhotel.com/reservations/123?persons=3>

- Cancel the reservation:

DELETE <http://myhotel.com/reservations/123>

# Best Practices for well-designed RESTful APIs

- Use only nouns for a URI:

~~/getAllReservations~~ GET /reservations

- Use plural nouns:

GET /reservations for all reservations

GET /reservations/123 for a specific reservation

- GET method should not alter the state of a resource
- Use sub-resources for relationships between resources

GET /reservations/123/persons/1: first occupant of the reservation #123

- Use “content-type” and “accept” HTTP headers to specify input/output format
- Provide proper HTTP status codes

# Best Practices for well-designed RESTful APIs

- Offer filtering and paging capabilities for large data sets

[GET /reservations?date=28/02/2021](#)

[GET /reservations?from=5&to=25](#)

- Version the API

2 strategies:

- In the URI: [GET /api/v2/reservations/123](#)



Easy to use with a web browser



Non-compliant with REST principle “one resource = one URI”

- In the accept header:

[GET /api/reservations/123 accept: application/v2](#)



More complex for the client



More REST-compliant