RESTFUL API
BEST PRACTICES

By Malwina Nowakowska

STX NEXT

talented developers | flexible teams | agile experts
Malwina Nowakowska

Developer

STX Next

Love Python
Special demo event
RESTful API
“Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web”
Architectural constraints

Client-Server

Stateless

Cacheable

Uniform Interface

Layered System

Code on Demand (optional)
Client-Server

Web app in Browser

Request

Internet

Response

Web Server

API

Database
Cacheable

Time

Server

No stored context on the server

Each request must contain all information

Client

Client has the right to reuse response data
Uniform Interface

1) Identifying the resource
2) Resource representation
3) Self-descriptive messages
4) Hypermedia as the engine of application state
Layered System

Each layer provides services to its neighbors.

Layers can encapsulate legacy services & protect new services from legacy clients.

Layers cannot "see" beyond their immediate neighbors.
Code on Demand (optional)

Add features to a deployed client, which provides for improved extensibility and configurability.

Better user-perceived performance and efficiency.
Architectural properties

Performance

Scalability

Simplicity of interfaces

Modifiability of components to meet changing needs (even while the application is running)

Visibility of communication between components by service agents

Portability of components by moving program code with the data

Reliability is the resistance to failure at the system level in the presence of failures within components, connectors, or data
RESTful APIs

Web service APIs that adhere to the REST architectural constraints are called RESTful APIs
Richardson REST Maturity Model

- Level 0: The Swamp of POX
- Level 1: Resources
- Level 2: HTTP Verbs
- Level 3: Hypermedia Controls

Glory of REST
/tickets
/events
/users
/baskets
/event_venues
/tickets
/events
/users
/baskets
/event_venues
/tickets
/events
/users
/baskets
/event_venues
Create POST
Read GET
Update PUT/PATCH
Delete DELETE
Create POST
Read GET
Update PUT/PATCH
Delete DELETE
GET /events - Read a list of events

GET /events/12 - Read a specific event

POST /events - Create a new event

PUT/PATCH /events/12 - Update event #12

DELETE /events/12 - Delete event #12
GET /events/12/prices - Read a list of event’s prices
GET /events/12/prices/5 - Read a specific event’s price
POST /events/12/prices - Create a new event’s price
PUT/PATCH /events/12/prices/5 - Update event’s price
DELETE /events/12/prices/5 - Delete event’s price
Related resource representations

```json
{
  "id": "123",
  "name": "Madonna",
  "prices": [
    {
      "name": "Circle",
      "value": "$5"
    },
    ...
  ]
}
```
405
Method Not Allowed
Filtering

GET /events?state=active&category=music,comedy
Sorting

GET /events?sort=-date,name
Searching

GET /events?q=Madonna
Limiting response

GET /events?fields=(id,title,date,artist.name)
Actions

GET /search

POST /order/1234/sum
Versioning

https://my-site.com/api/v1

Accept: application/vnd.my-site.v3+json
Pagination

Range: events=0-30

/events?page=1

/events?page=1&per_page=50 limit=10&offset=30
Accept-Ranges: events 50

Content-Range: 0-10/1234

offset = limit / count

resource max

Link:
<https://api.github.com/user/repos?page=3&per_page=100>; rel="next",
<https://api.github.com/user/repos?page=50&per_page=100>; rel="last"
Pretty print
HATEOAS

```json
{
    "id": "123",
    "name": "Madonna",
    ...
    "links": [
        {
            "rel": "self",
            "href": "https://api.events.com/v1/events/123",
            "method": "GET"
        },
        {
            "rel": "artists",
            "href": "https://api.events.com/v1/artists/123",
            "method": "GET"
        },
        ...
    ]
}
```
Best practices
Thank you!