

# Fullstack Development

# **API Design and Security**

# Content

- RESTful API Design
- API Security

# RESTful API Design

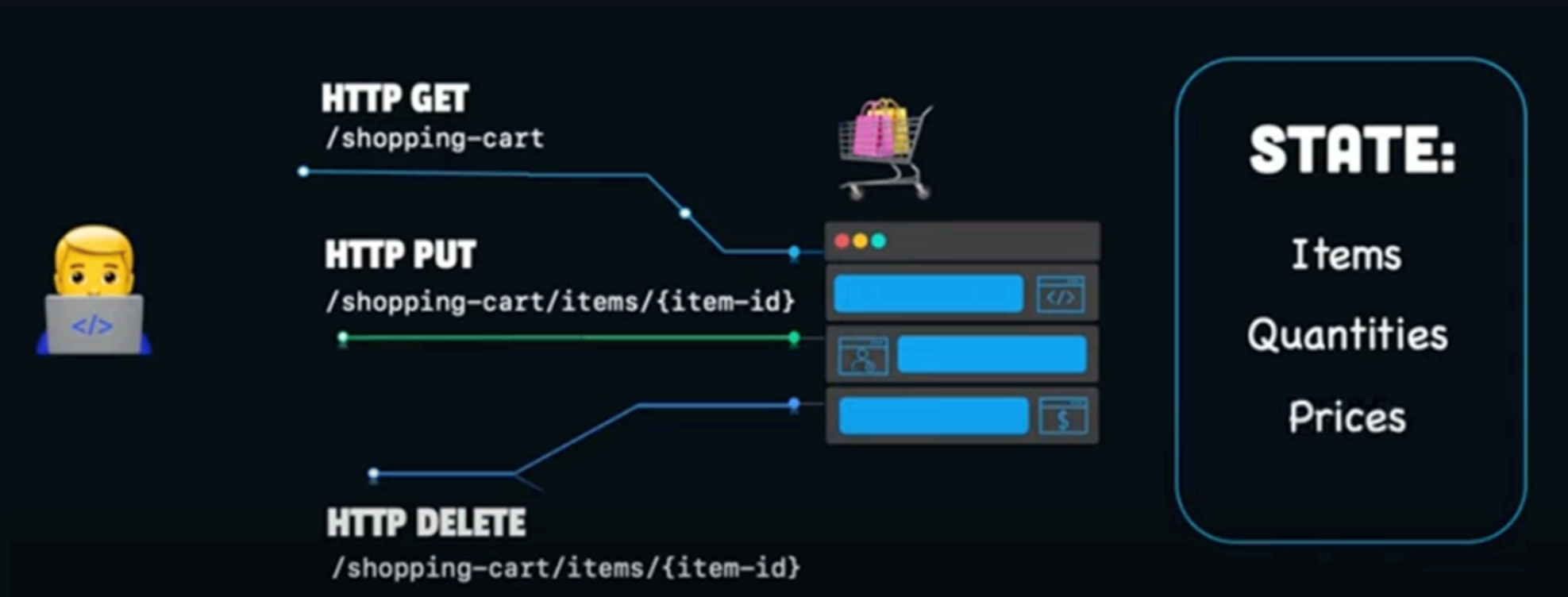
# RESTful API Design

## Stateless

- **Single-server** environment
  - Server process any request without knowledge of previous requests
- **Multi-server** environment
  - Request not bound to a specific server
  - Able to provide service in `load-balanced` fashion
- Make APIs more `scalable` > more available
- Easy to `cache` (without state info)

# RESTful API Design (2)

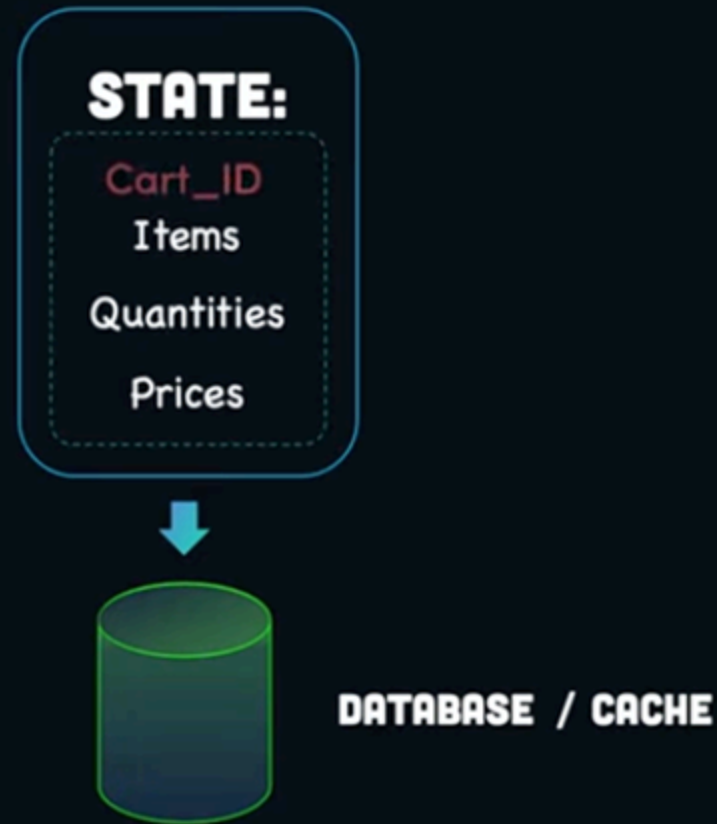
Most applications require some forms of state information



# RESTful API Design (3)

## 2. STORE THE STATE EXTERNALLY

### 2.1. USE UNIQUE IDENTIFIERS



# RESTful API Design (4)

3. USE IDS TO GET THE  
STATE OF THE CART



Cookie: Cart\_ID

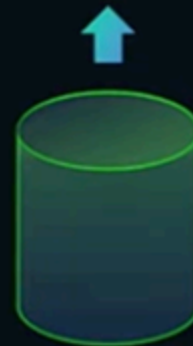
**STATE:**

Cart\_ID

Items

Quantities

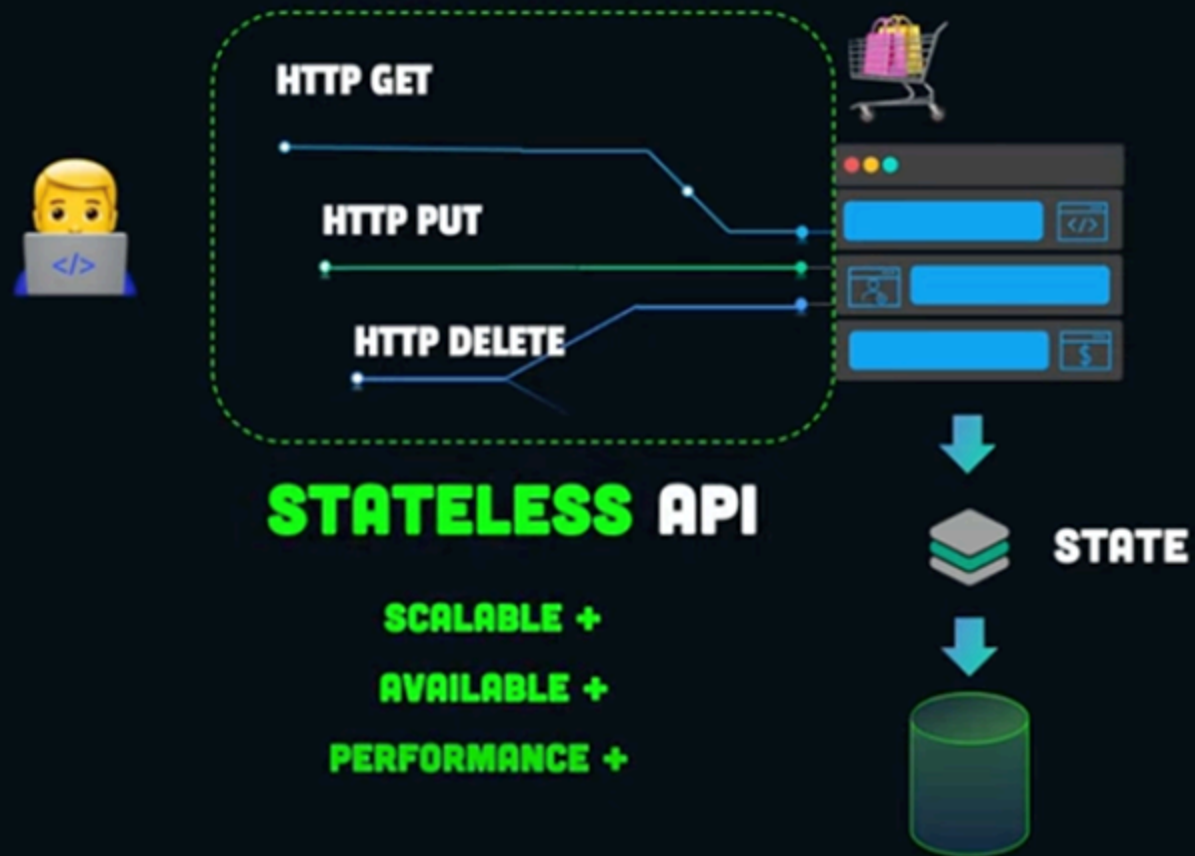
Prices



**DATABASE / CACHE**



# RESTful API Design (5)



# RESTful API Design (6)

Organize API around **resources** (not actions/verbs)

`https://e-commerce.com/orders` // Good → **RESOURCE**

`https://e-commerce.com/generate-order` // Avoid → **ACTION**

`https://e-commerce.com/users` // Good → **RESOURCE**

`https://e-commerce.com/create-user` // Avoid → **ACTION**

Using **HTTP method** to differentiate **actions**

# RESTful API Design (7)

Endpoint consistency is key, using plural form in URLs is recommended

Create order



+

<https://e-commerce.com/orders>

<https://e-commerce.com/initiateOrder>

<https://e-commerce.com/makeOrder>

<https://e-commerce.com/conceiveOrder>

<https://e-commerce.com/fatherOrder>

# RESTful API Design (8)

## HTTP Methods and CRUD operations



# RESTful API Design (9)

How to do complex query?


**Q:** GET ALL ORDERS MADE BY A CUSTOMER 

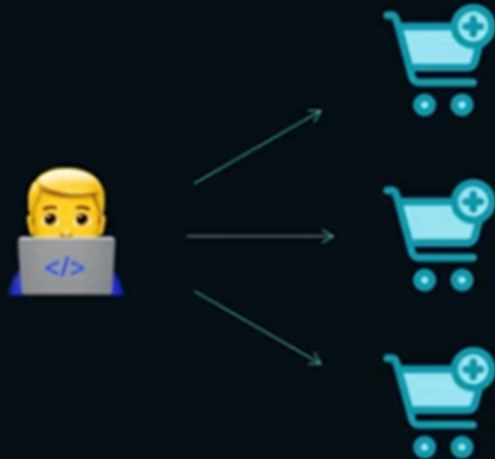
**+ SORTED**

**+ PAGINATED**

# RESTful API Design (10)

Entities are grouped into collections

**HTTP GET**  /customers/customer/orders



**Tip 1** Entities are grouped into collections.

-> intuitive API

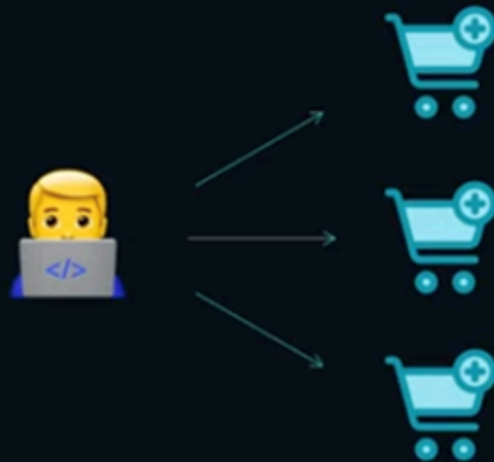
-> consistent naming convention

# RESTful API Design (11)

Use **parameterized URIs** for identity

`/customers/orders?customer_id=1` 🙄

**HTTP GET** `/customers/{customer_id}/orders`



**Tip 2** Use parametrized URIs for identity.

# RESTful API Design (12)

Avoid deeply nested URLs (`/collection/item/collection`)

**HTTP GET** `/customers/{customer_id}/orders`



**Tip 3** Avoid resource URLs more complex than `/collection/item/collection`

`/customers/1/orders/99/products` 👎

`/customers/1/orders`  
`/orders/99/products` 👍

GET `/customers/1?order=99`



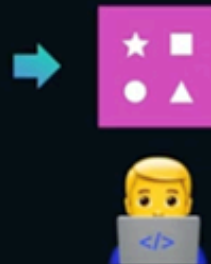
# RESTful API Design (14)

## Do not return plain-text response

- Client has to do some extra parsing/processing

HTTP GET /api/fruits

```
1 genus name id family order nutritions/carbohydrates nutritions/protein
2 nutritions/fat nutritions/calories nutritions/sugar Melus Apple 6 Rosaceae
3 Rosales 12.4 8.3 8.4 52 18.3 Prunus Arctostaphylos 10 Rosaceae Rosales 3.5 8.5
4 8.1 15 3.2 Prunus Avocado 88 Lauraceae Laurales 8.53 2 14.86 168 8.68
5 Musa Banana 1 Musaceae Zingiberales 22 1 8.2 96 17.2 Rubus Blackberry
6 64 Rosaceae Rosales 9 1.3 8.4 46 4.5 Fragaria Strawberry 31 Rosaceae
7 Rosales 5.5 8 8.4 29 5.4 Prunus Cherry 9 Rosaceae Rosales 12 1 8.3
8 18 8 Vaccinium Cranberry 87 Ericaceae Ericales 12.2 8.4 8.1 46 4
9 Solanum Dragonfruit 88 Solanaceae Caryophyllales 9 9 1.5 68 80rta
10 Durian 180 Palisade Malvaceae 27.1 1.5 1.5 147 6.75 Salicaceae Felice 76
11 Myrtaceae Myrtaceae 8 8.6 8.4 44 3Pinus Fig 68 Moraceae Rosales 18
12 8.6 8.3 74 18 Rides Gooseberry 60 Grossulariaceae Saxifragales 18 8.9 8.6
13 44 8 Vitis Grape 81 Vitaceae Vitales 18.1 8.72 8.36 68 14 Melus
14 Grossularia 72 Rosaceae Rosales 3.1 8.4 8.1 21 6.4 Psidium Guava 37
15 Myrtaceae Myrtales 14 3.4 3 68 8 Apteryx Kiwi 68 Actinidiaceae
16 Syzygium Kumquat 15 1.1 8.5 81 8 Actinidia Kiwifruit 85 Actinidiaceae
17 Ericales 14.8 1.14 8.5 81 8.9 Citrus Lemon 26 Rutaceae Sapindales 9
18 1.1 8.3 29 2.5 Citrus Lime 44 Rutaceae Sapindales 8.4 8.3 8.1 25 1.7
19 Vaccinium Lingonberry 65 Ericaceae Ericales 11.3 8.75 8.34 14 5.74
20 Litchi Lychee 67 Sapindaceae Sapindales 17 8.4 8.44 46 13 Mangifera
21 Mango 27 Anacardiaceae Sapindales 15 8.82 8.38 68 13.7 Cocos Melon 41
22 Cucurbitaceae Cucurbitaceae 8 8 8 34 8 Morus Mulberry 82 Moraceae Rosales
23 8.5 1.44 8.38 43 8.1 Citrus Orange 2 Rutaceae Sapindales 8.3 1 6.2 43
24 8.2 Carica Papaya 42 Caricaceae Caricaceae 11 8 8.4 43 3 Passiflora
25 Passiflora 76 Passifloraceae Malpighiales 22.4 2.2 8.7 97 11.2 Prunus
26 Peach 86 Rosaceae Rosales 9.5 8.5 8.25 29 8.4 Pyrus Pear 4 Rosaceae
27 Rosales 15 8.4 8.1 57 18 Diospyros Persimmon 50 Ebenaceae Rosales 18 8
28 8 81 18 Annona Pineapple 18 Bromeliaceae Poales 13.12 8.54 8.12 18
29 8.85 Cactaceae Pitahaya 76 Cactaceae Caryophyllales 7 1 8.4 36 3 Prunus
30 Plum 71 Rosaceae Rosales 13.4 8.7 8.26 46 9.52 Punicia Pomegranate 79
31 Lythraceae Myrtales 18.7 1.7 1.2 83 13.7 Rubus Raspberry 25 Rosaceae
32 Rosales 12 1.2 8.7 53 4.4 Fragaria Strawberry 3 Rosaceae Rosales 5.5
33 8.5 8.4 29 3.4 Citrus Tangerine 77 Rutaceae Sapindales 8.3 8.4 46 9.1
34 Solanum Tomato 5 Solanaceae Solanaceae 3.5 8.5 8.2 74 3.6 Citrus
35 Watermelon 25 Cucurbitaceae Cucurbitales 8 8.6 8.2 38 8
```



```
{
  {
    "genus": "Fragaria",
    "name": "Strawberry",
    "id": 3,
    "family": "Rosaceae",
    "order": "Rosales",
    "nutritions": {
      "carbohydrates": 5.5,
      "protein": 0,
      "fat": 0.4,
      "calories": 29,
      "sugar": 5.4
    }
  },
  {
    "genus": "Musa",
    "name": "Banana",
    "id": 2,
    "family": "Musaceae",
    "order": "Zingiberales",
    "nutritions": {
      "carbohydrates": 22,
      "protein": 0,
      "fat": 0.2,
      "calories": 96,
      "sugar": 17.2
    }
  }
}
```

# RESTful API Design (15)

Response with **JSON** / **XML** / **YAML**



JSON

```
{  
  "configurations": [  
    {  
      "name": "Ingress",  
      "value": "data/input"  
    },  
    {  
      "name": "Egress",  
      "value": "data/output"  
    }  
  ]  
}
```



XML

```
<Configurations>  
  <Config>  
    <Name>Ingress</Name>  
    <Value>data/input</Value>  
  </Config>  
  <Config>  
    <Name>Egress</Name>  
    <Value>data/output</Value>  
  </Config>  
</Configurations>
```



YAML

```
---  
configurations:  
- name: Ingress  
  value: data/input  
- name: Egress  
  value: data/output
```

# RESTful API Design (16)

Allow client to specify **Content-Type** header

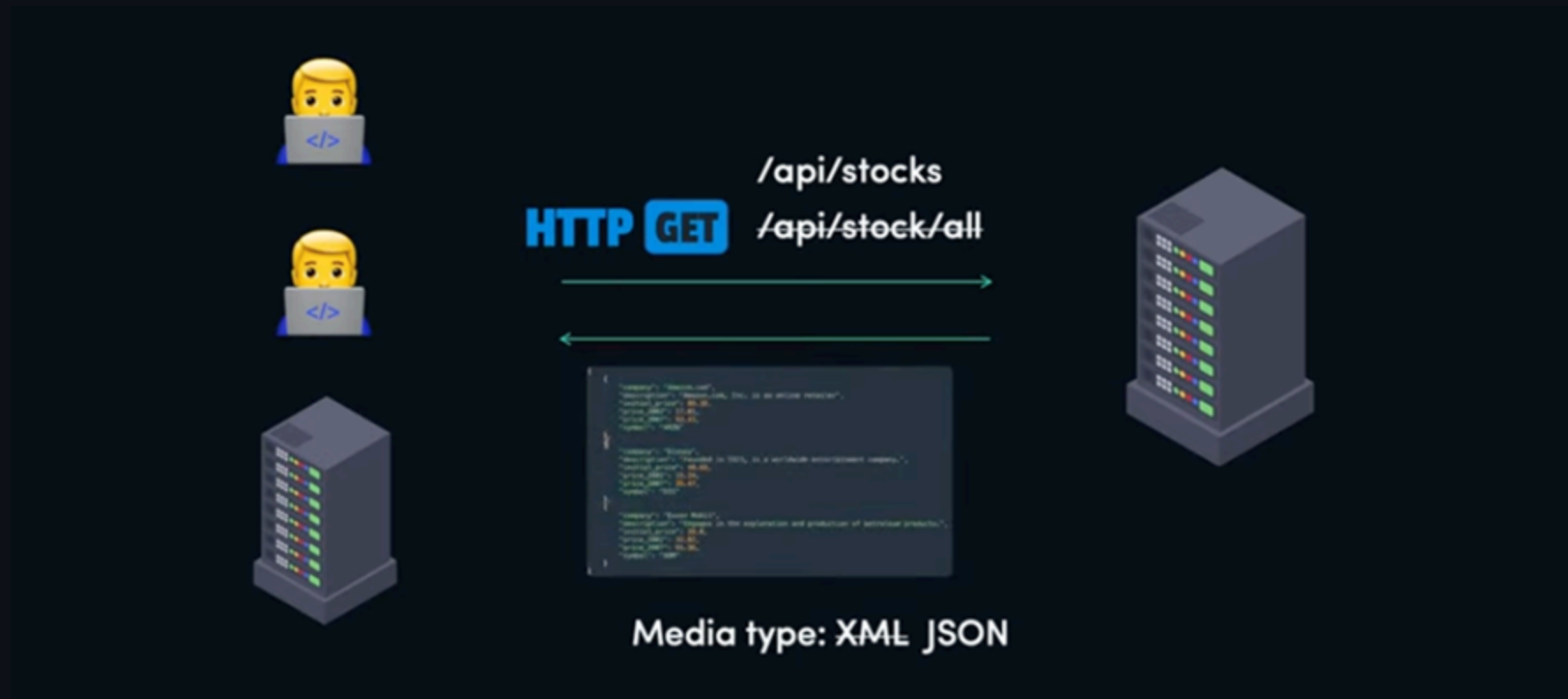
**HTTP** **Header**

**Content-Type : application/json**



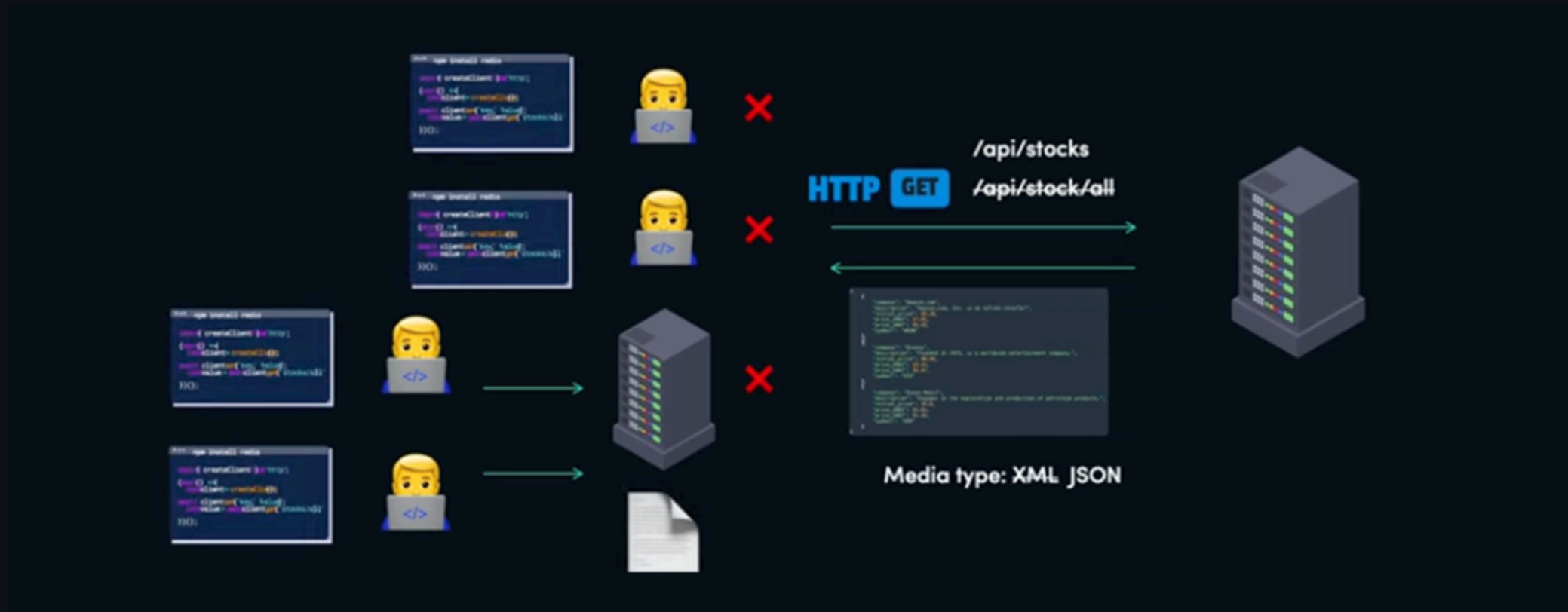
# RESTful API Design (17)

Changing API after it has been adopted is dangerous!



# RESTful API Design (18)

Clients need to **update code/documents** to make app work again



# RESTful API Design (19)

However, business requirements are always changed

**HTTP GET** /api/stocks

Add new field  
→

Change field to list  
→

Change media type  
→

etc

```
{  
  "company": "Amazon.com",  
  "description": "Amazon.com, Inc. is an online retailer",  
  "initial_price": 89.38,  
  "price_2002": 17.01,  
  "price_2007": 93.43,  
  "symbol": "AMZN"  
},  
  {  
    "company": "Disney",  
    "description": "Founded in 1923, is a worldwide entertainment company.",  
    "initial_price": 40.60,  
    "price_2002": 15.24,  
    "price_2007": 35.47,  
    "symbol": "DIS"  
  },  
  {  
    "company": "Exxon Mobil",  
    "description": "Engages in the exploration and production of petroleum products.",  
    "initial_price": 39.0,  
    "price_2002": 32.02,  
    "price_2007": 91.36,  
    "symbol": "XOM"  
  }  
}
```

# RESTful API Design (20)

API versioning is the answer

## API VERSIONING

### URI

`https://e-commerce.com/v2/customers/`

### QUERY PARAMS

`https://e-commerce.com/customers?version=2`

### HEADERS

`https://e-commerce.com/customers`

`Custom-Header: api-version=2`

### MEDIA TYPE

`https://e-commerce.com/customers`

`Accept: application/e-commerce.v1+json`

# RESTful API Design (21)

## API versioning comparison

	CACHE-FRIENDLY	RESTFUL
URI	✓	
QUERY PARAMS	✓	
HEADERS		✓
MEDIA TYPE		✓



# RESTful API Design (22)

API exception handling and response with proper HTTP status code



# RESTful API Design (23)

## HTTP status code



### 4xx

- 400** Bad Request
- 401** Unauthorized
- 403** Forbidden
- 404** Not Found
- 405** Method Not Allowed
- 406** Not Acceptable
- 404** Not Found
- 409** Conflict
- 423** Locked



### 5xx

- 500** Internal Server Error
- 503** Service Unavailable



# RESTful API Design (24)

Including links in JSON responses is a common practice

```
{
  "id": 1,
  "name": "Example",
  "links": {
    "self": "http://api.example.com/resource/1",
    "related": "http://api.example.com/resource/2"
  }
}
```

# RESTful API Design (25)

## Content return policies

Return the created resource

- HTTP status code: 201 Created, Location header of resource's URI
- Full detail of the newly created resource

```
HTTP/1.1 201 Created
Location: /resources/123
Content-Type: application/json
{
  "id": 123,
  "name": "New Resource"
}
```

# RESTful API Design (26)

## Content return policies

### Do not return content

- Only a `201 Created` or `204 No Content` response, and `Location` header
- Minimizes data transfer

```
HTTP/1.1 201 Created  
Location: /resources/123
```

# References

- [Best practices for REST API design](#)
- [Mastering RESTful API Design: A Practical Guide](#)
- [REST API - Best Practices - Design](#)