

# **HTTP API Design: A Beginner's Guide**

**Janani Subbiah**

# Who am I?

- I am Janani!
- Staff Engineer, Detroit Labs
- Love to read and travel
- Balcony gardening!

# Agenda

- API and ReST
- API Design
- 6-step Process to HTTP API Design
- Next steps
- Conclusion

# What is an API?

Application Programming Interface

Exposes functionality

Examples: Collections from Java's Utils library, APIs by OpenWeatherMap

# What is ReST?

Representational State Transfer

Architectural concept

Client–server

Cacheable

Uniform Interface

Stateless

Layered system

Introduced by Roy Fielding in 2000

Allows building scalable, distributed  
systems

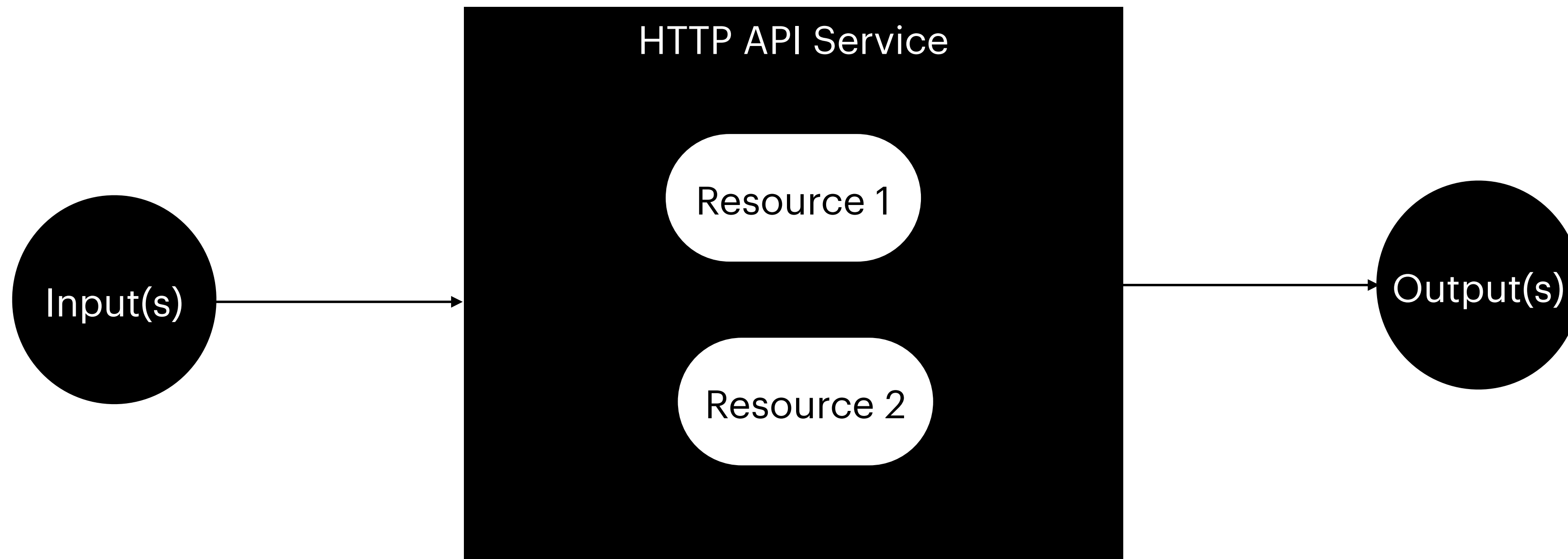
# ReST & HTTP

ReST does not dictate protocol

HATEOS: Hypermedia As The Engine of Application State

Today, we are going to be building an **HTTP API Service**

# HTTP API Design: What?



# HTTP API Design: For?

## Developers

- Building the API
- Consuming the API

## Other roles

- Designers
- Managers
- Quality Engineers etc



# HTTP API Design: When?

- Starts pre-development
- Continues alongside development work
- Iteratively improved

# 6-step Process to HTTP API Design

- API Naming
- HTTP Verbs
- Request Body
- Response Status
- Response Body
- Path vs Query Params

# Step 1: API Naming

# API Naming

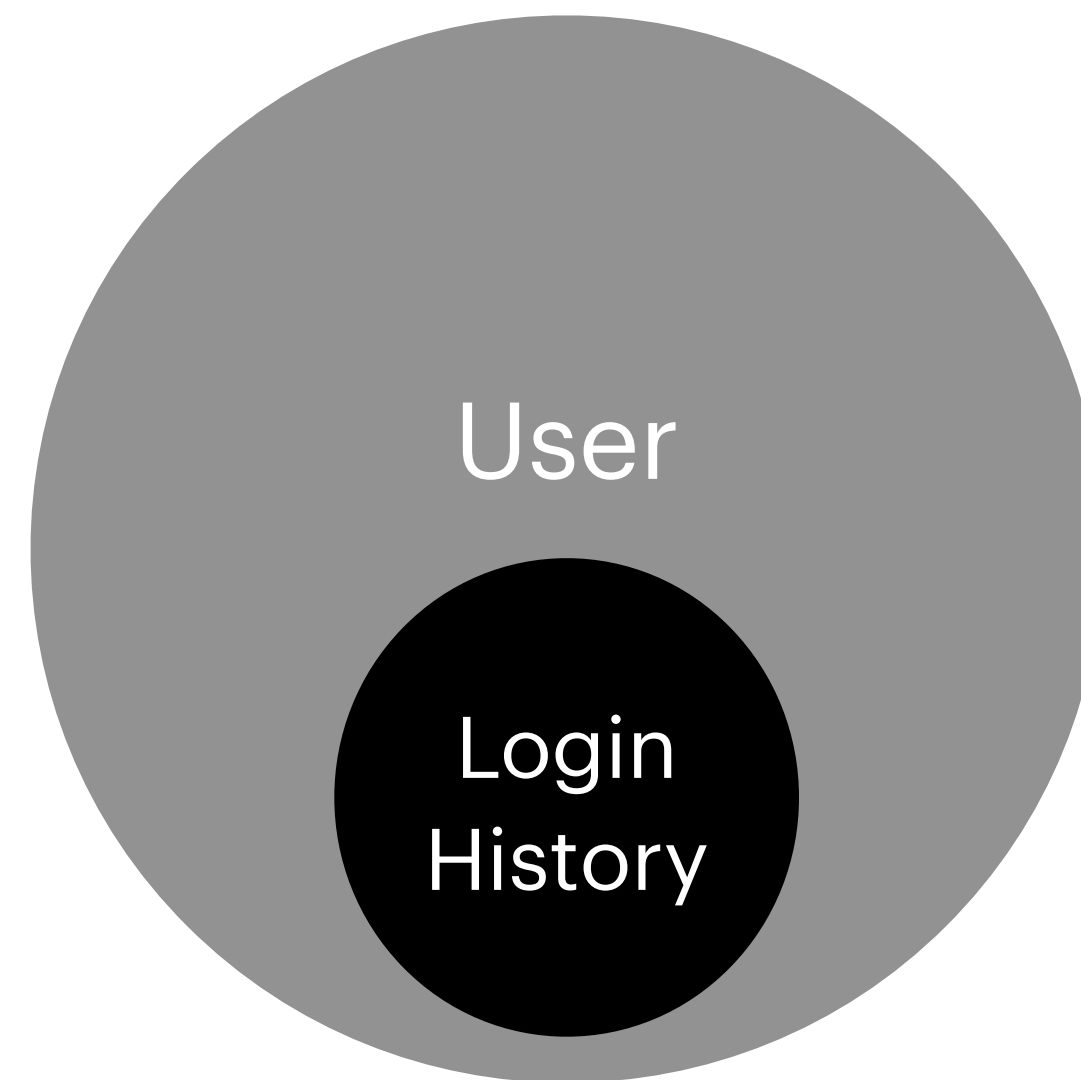
## Identify the resource

- What is this API for?
- Does it belong to another resource?

## Consistent URI naming

- Camel case
- Hyphenated

# API Naming: Example



`/users/{userId}/loginHistories`

`/users/{userId}/loginHistories/{loginHistoryId}`

# 6-step Process to HTTP API Design

- ~~API Naming~~
- HTTP Verbs
- Request Body
- Response Status
- Response Body
- Path vs Query Params

# Step 2: HTTP Verbs

# HTTP Verbs

## Operations

- C: Create (POST)
- R: Read (GET)
- U: Update (PUT)
- D: Delete (DELETE)

Identify allowed operations

Pick the Verb!



# HTTP Verbs: HEAD & PATCH

## HEAD

~ GET

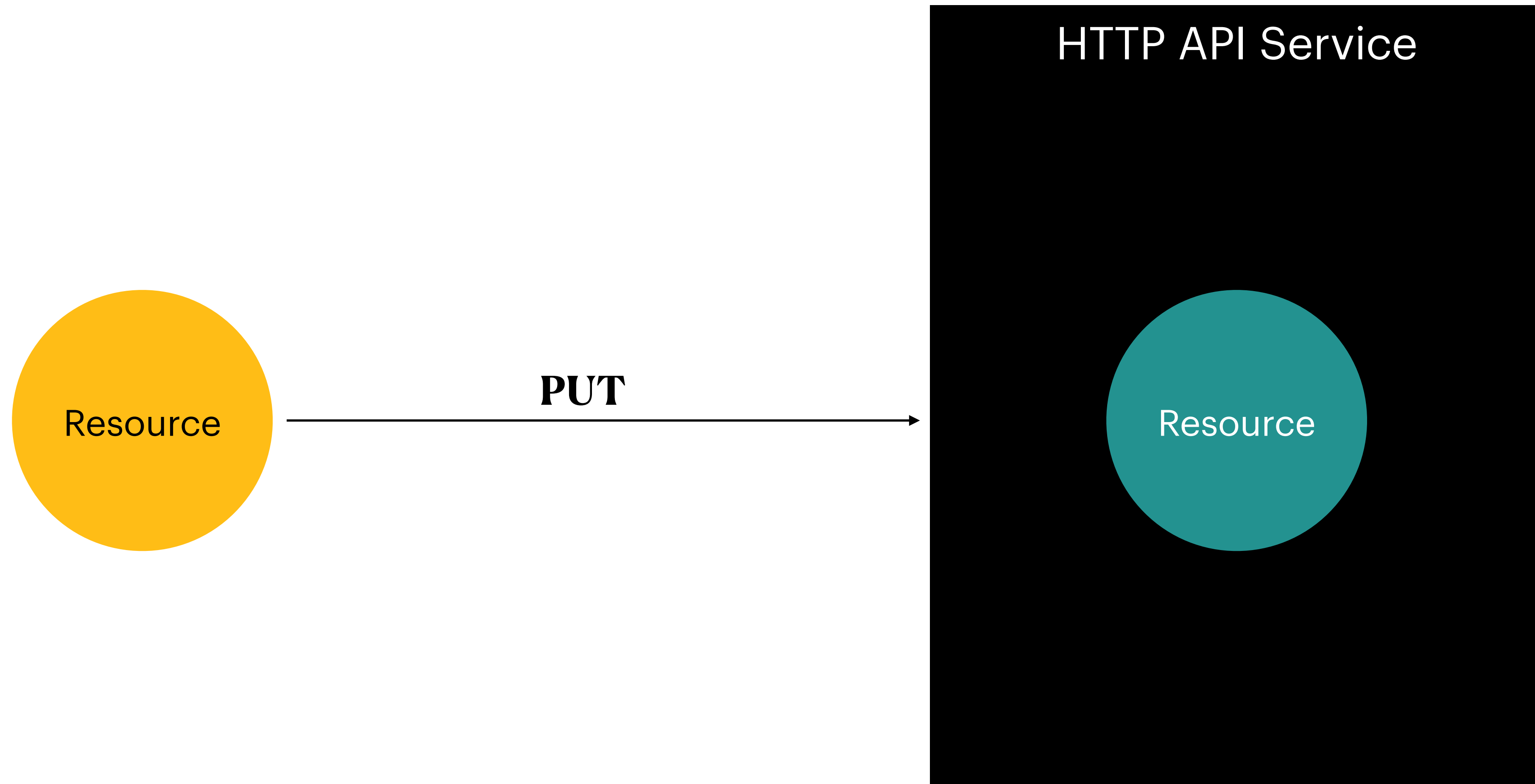
Metadata about the data

## PATCH

~ PUT

Modifies the resource

# HTTP Verbs: PUT vs PATCH



# HTTP Verbs: PUT vs PATCH

Field name: Square  
Operation: Update

New  
Value

**PATCH**

HTTP API Service

Old  
Value

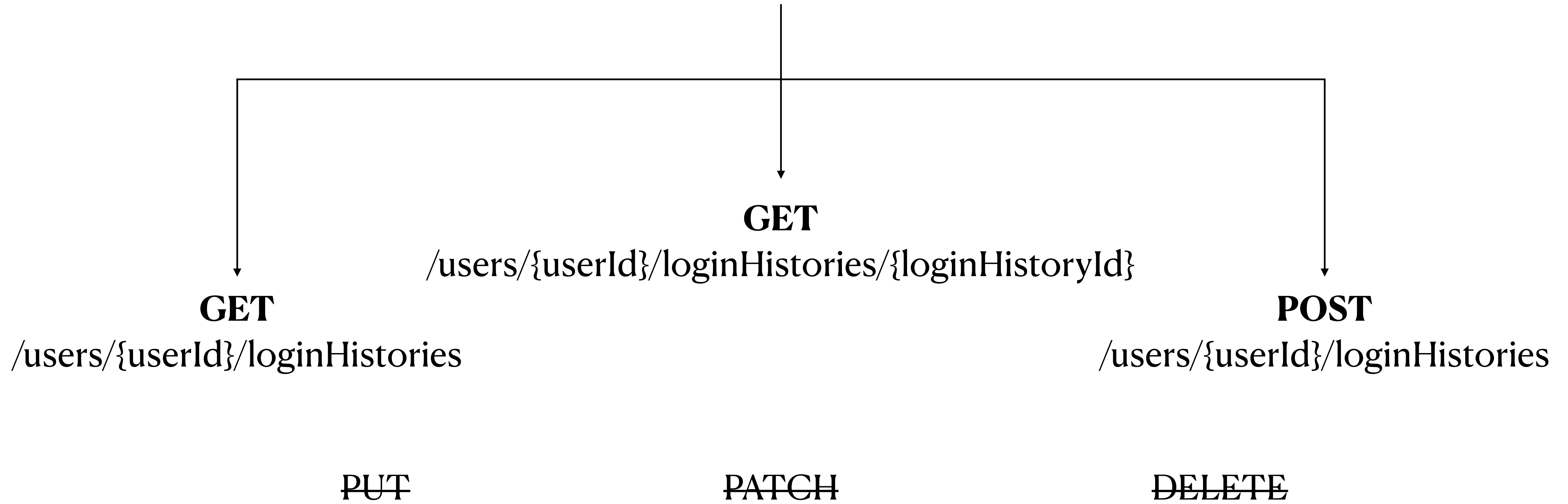


# HTTP Verbs: POST

Can be used for computationally intense operations!

# HTTP Verbs: Example

Immutable: Only read & create



# 6-step Process to HTTP API Design

- ~~API Naming~~
- ~~HTTP Verbs~~
- Request Body
- Response Status
- Response Body
- Path vs Query Params

# Step 3: Request Body

# Request Body

Is request body needed?

RFC recommendations

Format (JSON/XML/Text)

Request body structure

- What is needed to perform the operation
- Validations
- Avoid duplicating request components



# Request Body: Example (GET)

**GET**

/users/{userId}/loginHistories

/users/{userId}/loginHistories/{loginHistoryId}

Does the operation need a request body? **No**

# Request Body: Example (GET)

~~GET~~

~~/users/{userId}/loginHistories~~

~~/users/{userId}/loginHistories/{loginHistoryId}~~

# Request Body: Example (POST)

**POST**

|

`/users/{userId}/loginHistories`

Does the operation need a request body? **Yes**

# Request Body: Example (POST)

POST

|

/users/{userId}/loginHistories

Does the RFC recommend it? **Yes**

# Request Body: An example (POST)

**POST**

|

`/users/{userId}/loginHistories`

Format: **Json**

# Request Body: An example (POST)

**POST**

|

`/users/{userId}/loginHistories`

{

**“device”: “iPhone”,**

**“time”: “1608739241”,**

**“registeredDevice”: true,**

**“ipAddress”: “000.000.00.000”,**

**“location”: “Detroit, MI, USA”**

}

# 6-step Process to HTTP API Design

- ~~API Naming~~
- ~~HTTP Verbs~~
- ~~Request Body~~
- Response Status
- Response Body
- Path vs Query Params

# **Step 4: Response Status**



# Response Status

- Did the operation succeed?
- Next steps (if any)?

# Response Status

## Error Status Code

4xx

5xx

## Success Status Code

200

201

204

# Response Status: Example (Error)

**POST**    /users/{userId}/loginHistories

{

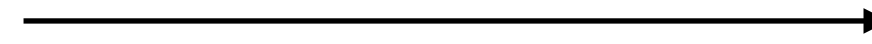
  “time”: “1608739241”,

  “registeredDevice”: true,

  “ipAddress”: “000.000.00.000”,

  “location”: “Detroit, MI, USA”

}



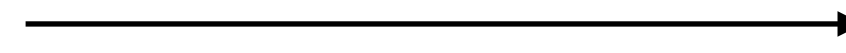
HTTP API Service

400



# Response Status: Example (Error)

**GET** /users/no-user-id/loginHistories

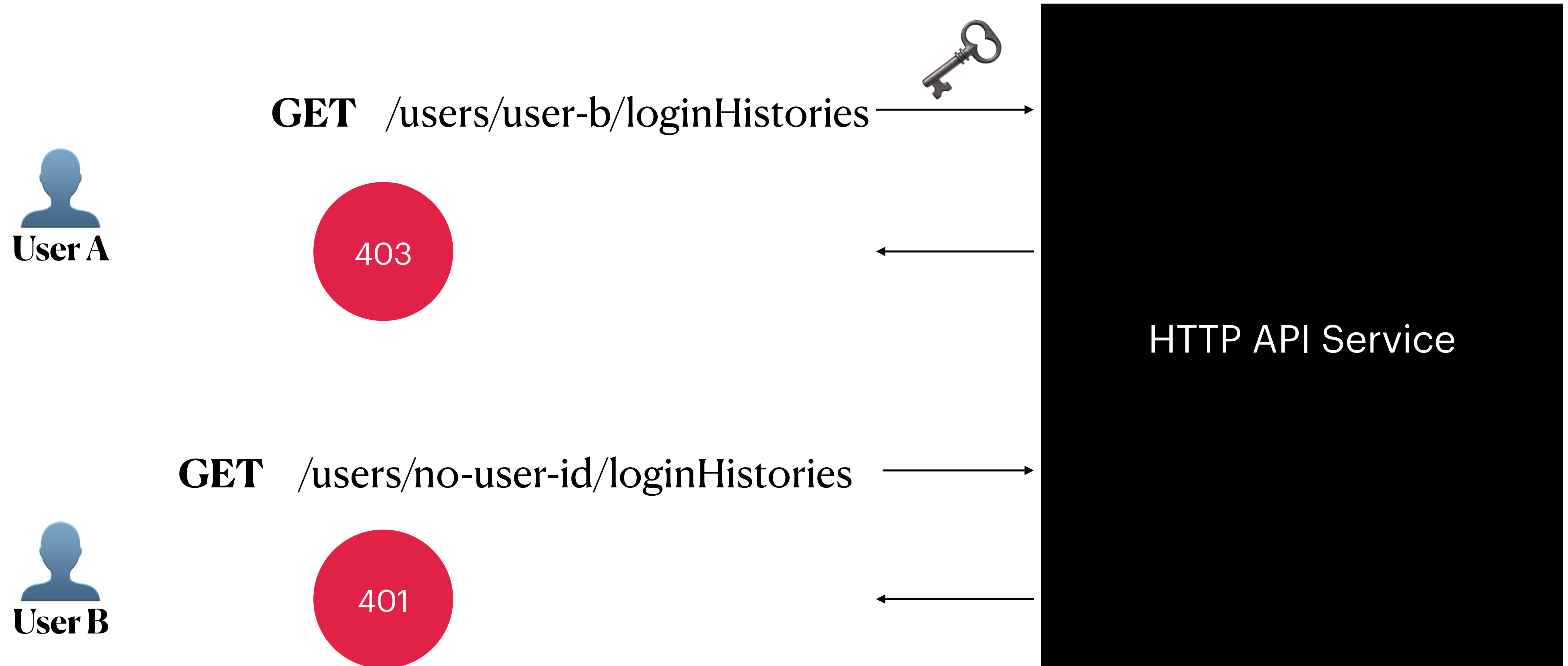


HTTP API Service

404



# Response Status: Example (Error)



# Response Status: Example (Success)



# Response Status: Example (Success)



**POST** /users/user-a/loginHistories

{

“device”: “iPhone”,

“time”: “1608739241”,

“registeredDevice”: true,

“ipAddress”: “000.000.00.000”,

“location”: “Detroit, MI, USA”

}

201

HTTP API Service

# 6-step Process to HTTP API Design

- ~~API Naming~~
- ~~HTTP Verbs~~
- ~~Request Body~~
- ~~Response Status~~
- Response Body
- Path vs Query Params



# Step 5: Response Body

# Response Body

- Should be on par with the status
- Decide what informations needs to be sent back
- Break up responses (even if it means additional APIs)

# Response Body: Example (GET ALL)

```
[  
  {  
    "id": "1",  
    "device": "iPhone",  
    "time": "1608739241"  
  },  
  {  
    "id": "2",  
    "device": "iPhone",  
    "time": "1608739241"  
  },  
  ...  
]
```

# Response Body: Example (GET ONE)

```
{
```

```
  "id": "1",
```

```
  "device": "iPhone",
```

```
  "time": "1608739241",
```

```
  "registeredDevice": true,
```

```
  "ipAddress": "000.000.00.000",
```

```
  "location": "Detroit, MI, USA"
```

```
}
```

# Response Body: An example (POST)

```
{
```

```
  "device": "iPhone",
```

```
  "time": "1608739241",
```

```
  "registeredDevice": true,
```

```
  "ipAddress": "000.000.00.000",
```

```
  "location": "Detroit, MI, USA"
```

```
}
```

# 6-step Process to HTTP API Design

- ~~API Naming~~
- ~~HTTP Verbs~~
- ~~Request Body~~
- ~~Response Status~~
- ~~Response Body~~
- Path vs Query Params

# Step 6: Path vs Query Params

# Path vs Query Parameter

- Path Parameter
  - Unique resource identifiers
  - Nested appropriately
- Query Parameter
  - Search
  - Sort
  - Pagination



# Path vs Query Parameter: Example

Path Variable



Unique Identifier



**GET** /users/{userId}/loginHistories

**GET** /users/{userId}/loginHistories/{loginHistoryId}

# Path vs Query Parameter: Example

## Query Parameters

Search

**GET** /users/{userId}/loginHistories?device= iPhone

Sort

**GET** /users/{userId}/loginHistories?asc= time

**GET** /users/{userId}/loginHistories?desc= time

Pagination

**GET** /users/{userId}/loginHistories?page=2

# 6-step Process to HTTP API Design

- ~~API Naming~~
- ~~HTTP Verbs~~
- ~~Request Body~~
- ~~Response Status~~
- ~~Response Body~~
- ~~Path vs Query Params~~

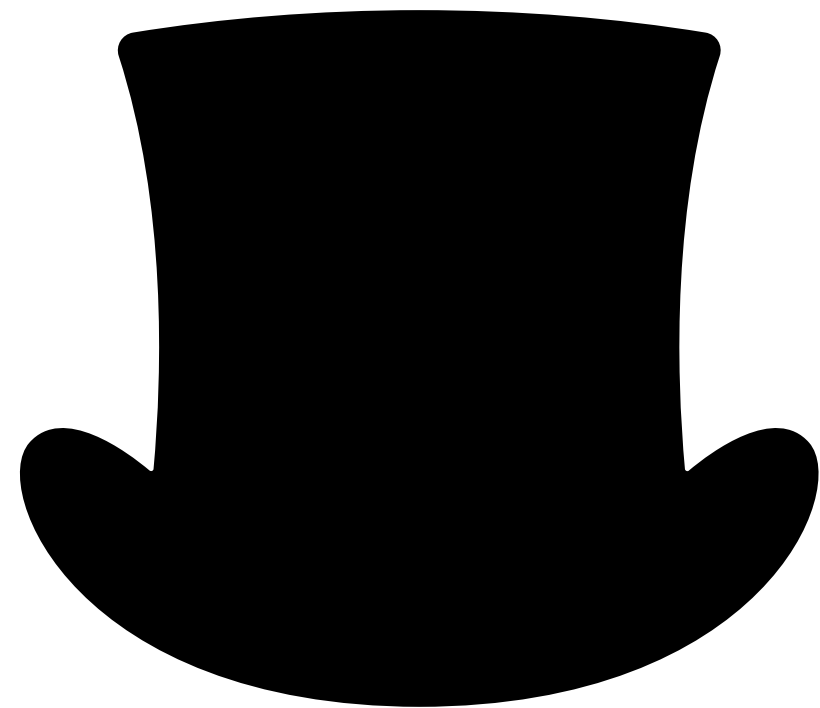
**What is next?**

# Security



- Secure APIs: Username Password, Json Web Tokens etc
- API Keys
- Compliance (E.g.: Payment Card Industry aka PCI)
- Open Worldwide Application Security Project (OWASP)

# Headers



- Pre-defined headers
- Consistent header names!

# Versioning

✖.✖.✖

- In the URL (Path variable/Query Param)
- Header (existing or custom)

Do you need to version your service?

# Conclusion

- API, ReST & API Design
- 6-Step Process
  - API Naming
  - HTTP Verbs
  - Request Body
  - Response Status
  - Response Body
  - Path vs Query parameters
  - Next Steps: Security, Headers and Versioning
- The RFC is your friend (<https://tools.ietf.org/html/rfc7231>)



# Thank you!

