RESTful API

Seyed Mohammad Mousavi

RESTful API

REST: Representational State Transfer

API: Application Programming Interface

Interface

What is Interface?

Interface is Shared Boundary to exchange information.

This Exchange could be Between:

- Different parts of a software
- Two different software
- Software and hardware
- Software and human

API

API is the interface for exchange data between two software.

Why should we write an API for our Software?

- To develop Multiple Front-Ends for our Back-End
- To let others create their own program with our API
- To let others use our API within their program

RESTful API

REST is a software architectural style that defines a set of constraints to be used for creating Web Services and it usually uses HTTP.

When HTTP is used, as is most common, the operations available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE.

In a RESTful Web Services, requests made to a resource's URI will elicit a payload formatted in HTML, XML, JSON or other formats.

REST Properties

The constraints of the REST architectural affect the following properties:

- Performance
- Scalability
- Simplicity of a uniform interface
- Modifiability of components to meet changing needs
- Portability of components

REST Constraints

Six guiding constraints define a RESTful system.

Theses constraints restrict the ways that the server can process and respond to client requests.

By operating within these constraints, the system gains desirable non-functional properties, such as performance, scalability and …

If a system violates any of the required constraints, it cannot be considered RESTful.

REST Constraints 1: Client–server architecture

The principle behind the client–server constraints is the separation of concerns. Separating the user interface concerns from the data storage concerns improves the portability of the user interfaces across multiple platforms.

It also improves scalability by simplifying the server components.

Perhaps most significant to the Web is that the separation allows the components to evolve independently

REST Constraints 2: Statelessness

Servers don’t hold the state.

State should be transferred to the server within the request.

REST Constraints 3: Cacheability

As on the World Wide Web, clients and intermediaries can cache responses.

Responses must, implicitly or explicitly, define themselves as either cacheable or non-cacheable.

REST Constraints 4: Layered System

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

If a proxy or load balancer is placed between the client and server, it won't affect their communications and there won't be a need to update the client or server code.

Intermediary servers can improve system scalability by enabling load balancing and by providing shared caches.

REST Constraints 5: Code on Demand

This constraint is optional.

Servers can temporarily extend or customize the functionality of a client by transferring executable code: for example, client-side scripts such as JavaScript.

REST Constraints 6: Uniform Interface

The uniform interface constraint is fundamental to the design of any RESTful system.

It simplifies and decouples the architecture, which enables each part to evolve independently.

The four constraints for this uniform interface are:

- Resource identification in requests
- Resource manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state

Example

An endpoint URL: https://university.com/student

GET: List of all students

```
[
  {
    "id": 29,
    "name": "Mohammad",
  },
  {
    "id": 30,
    "name": "Mina",
  }
]
```

Example

An endpoint URL: https://university.com/student/29

GET: Information about student with id 29

```
[
  {
    "id": 29,
    "name": "Mohammad",
  }
]
```

Example

An endpoint URL: https://university.com/student

POST: Create a new student

```
[
  {
    "id": 31,
    "name": "Ahmad",
  }
]
```

What is GraphQL?

GraphQL is a query language for your API.

Instead of sending GET request and get all the information that the API will send you, you send a query and receive only what you asked for.

Why bother? We can just use the information we need.

GraphQL Example

Instead of sending a GET and receive all the information we can ask just for what we need, for example:

```
{ student {name} }
```

```
[
  {
    "name": "Mohammad",
  },
  {
    "name": "Mina",
  }
]
```

An alternate for GraphQL

We can create different URIs for different needs of information.

Is this method better than using GraphQL?

Why JSON?

We can use any format for transferring our data wit RESTful API, Why JSON?

Because it’s natively supported in JavaScript

But is it the best format we can use?