Understanding REST

REST (Representational State Transfer) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. REST is an architectural style for designing distributed systems. It is not a standard but a set of constraints, such as being stateless, having a client/server relationship, and a uniform interface. REST is not strictly related to HTTP, but it is most commonly associated with it.

Principles of REST

- **Resources** expose easily understood directory structure URIs.
- **Representations** transfer JSON or XML to represent data objects and attributes.
- **Messages** use HTTP methods explicitly (for example, GET, POST, PUT, and DELETE).
- **Stateless** interactions store no client context on the server between requests. State dependencies limit and restrict scalability. The client holds session state.

HTTP methods

Use HTTP methods to map CRUD (create, retrieve, update, delete) operations to HTTP requests.

GET

Retrieve information. GET requests must be safe and idempotent, meaning regardless of how many times it repeats with the same parameters, the results are the same. They can have side effects, but the user doesn't expect them, so they cannot be critical to the operation of the system. Requests can also be partial or conditional.

Retrieve an address with an ID of 1:

```
GET /addresses/1
```

POST

Request that the resource at the URI do something with the provided entity. Often POST is used to create a new entity, but it can also be used to update an entity.

Create a new address:

```
POST /addresses
```

PUT

Store an entity at a URI. PUT can create a new entity or update an existing one. A PUT request is idempotent. Idempotency is the main difference between the expectations of PUT versus a POST request.

Modify the address with an ID of 1:

```
PUT /addresses/1
```

> **Note:** PUT replaces an existing entity. If only a subset of data elements are provided, the rest will be replaced with empty or null.

PATCH

Update only the specified fields of an entity at a URI. A PATCH request is idempotent. Idempotency is the main difference between the expectations of PUT versus a POST request.

```
PATCH /addresses/1
```

DELETE

Request that a resource be removed; however, the resource does not have to be removed immediately. It could be an asynchronous or long-running request.

Delete an address with an ID of 1:

```
DELETE /addresses/1
```

HTTP status codes

Status codes indicate the result of the HTTP request.

- **1XX** - informational
- **2XX** - success
- **3XX** - redirection
- **4XX** - client error
- **5XX** - server error

Media types

The `Accept` and `Content-Type` HTTP headers can be used to describe the content being sent or requested within an HTTP request. The client may set `Accept` to `application/json` if it is requesting a response in JSON. Conversely, when sending data, setting the `Content-Type` to `application/xml` tells the client that the data being sent in the request is XML.