Full Stack Application Development- SE ZG503

Akshaya Ganesan

CSIS, WILP

BITS Pilani

Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Lecture No: 8 REST

Module 4: Server Side: Implementing Web Services

innovate achieve lead

- REST
- Principles of REST
- REST constraints
- Service Design with REST
  - Interaction Design with HTTP
  - Interface Design (URI)
  - Representation and Metadata design
- Implementing REST API
  - Using a Framework
  - URL Mapping
  - Routing Requests-Redirection
  - Implementing a web server
  - Processing request, response, data
- Storing data in databases
  - Models
  - Object Relational Mapper
  - Interaction with DBs
- API versioning and documentation
-

innovate achieve lead

BITS Pilani

Pilani Campus

REST Services

REST - Concepts

- REST stands for REpresentational State Transfer
- REST is an architectural style.

- A RESTful web service
  - ✓ exposes information about itself in the form of information about its resources
  - ✓ enables the client to take actions or perform CRUD operations on those resources.

- Resource
  - ✓ Can be anything(docs, data, media, images) the web service can provide information about
  - ✓ Each resource has a unique identifier - can be a name or a number

When a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource!

REST - example

- When a developer calls Instagram API to fetch a specific user (the resource)
  - ✓ the API will return the state of that user, including
    - their name
    - the number of posts that user posted on Instagram so far
    - how many followers they have
    - and more

- The representation of the state can be in a JSON format
  - ✓ probably for most APIs this is indeed the case
  - ✓ but can also be in XML or HTML format

Resources

- The main building blocks of a REST architecture are the resources
- **Representations:** It can be any way of representing data (binary, JSON, XML, etc.).
- A single resource can have multiple representations.
- **Identifier:** A URL that retrieves only one specific resource at any given time.
- **Metadata:** Content type, last-modified time, and so forth.
- **Control data:** Is-modifiable-since, cache-control.

REST and HTTP

- The fundamental principle of REST is to use the **HTTP** protocol for data communication.
- RESTful web service makes use of HTTP for determining the action to be carried out on the particular resources
- REST gets its motivation from HTTP. Therefore, it can be said as a structural pillar of the REST
- Commonly Used ones
  - ✓ GET - to read (or retrieve) a representation of a resource
  - ✓ POST - utilized to create new resources
  - ✓ PUT - to update a resource
  - ✓ PATCH - to modify a resource - only needs to contain the changes to the resource, not the complete resource
  - ✓ DELETE - to delete a resource identified by a URI.

- Rarely Used ones
  - ✓ Head – requests the headers.
  - ✓ Options - requests permitted communication options for a given URL or server

REST

- Server responds based on the identifier and the operation
  - ✓ An identifier for the resource you are interested in
    - URL for the resource, also known as the endpoint
    - URL stands for Uniform Resource Locator
    - A REST service exposes a URI for every piece of data the client might want to operate on.(Scoping Information)
  - ✓ The operation you want the server to perform on that resource
    - in the form of an HTTP method, or verb
    - common HTTP methods are GET, POST, PUT, and DELETE
    - One way to convey method information in a web service is to put it in the HTTP method(Method Information)

Method Information

- In a SOAP, REST, RPC
- How the client can convey its intentions to the server?

REST Get Request

An HTTP GET request for http://www.oreilly.com/index.html

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,...
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The response to an HTTP GET request for http://www.oreilly.com/index.html

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
Server: Apache
Last-Modified: Fri, 17 Nov 2006 09:05:32 GMT
Etag: "7359b7-a7fa-455d8264
Accept-Ranges: bytes
Content-Length: 43302
Content-Type: text/html
X-Cache: MISS from www.oreilly.com
Keep-Alive: timeout=15, max=1000
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
...
<title>oreilly.com -- Welcome to O'Reilly Media, Inc.</title>
-----
```

Image Ref: Book T2: Restful Web services *Leonard Richardson and Sam Ruby*

SOAP Request

A sample SOAP RPC call

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<gs:doGoogleSearch xmlns:gs="urn:GoogleSearch">
<q>REST</q>
...
</gs:doGoogleSearch>
</soap:Body>
</soap:Envelope>
```

Part of the WSDL description for Google's search service

```
<operation name="doGoogleSearch">
<input message="typens:doGoogleSearch"/>
<output message="typens:doGoogleSearchResponse"/>
</operation>
```

XML-RPC Example

```
POST /rpc HTTP/1.1
Host: www.upcdatabase.com
User-Agent: XMLRPC::Client (Ruby 1.8.4)
Content-Type: text/xml; charset=utf-8
Content-Length: 158
Connection: keep-alive
<?xml version="1.0" ?>
<methodCall>
<methodName>lookupUPC</methodName>
...
</methodCall>
```

Example 1-12. An XML document describing an XML-RPC request

```
<?xml version="1.0" ?>
<methodCall>
<methodName>lookupUPC</methodName>
<params>
<param><value><string>001441000055</string></value
></param>
</params>
</methodCall>
```

Scoping Information

- *http://flickr.com/photos/tags/penguin*
  - *http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=penguin*
- http://www.upcdatabase.com/upc/00598491'

- One obvious place to put it is in the URI path.

- A RESTful, resource-oriented service exposes a URI for every piece of data the client might want to operate on.

Method and Scoping information

- In RESTful web service,
  - ✓ the method information goes into the HTTP method.
  - ✓ The scoping information goes into the URI.
- Given the first line of an HTTP request to a RESTful web service you should understand basically what the client wants to do.
- “GET /reports/docs HTTP/1.1”
- If the HTTP method doesn’t match the method information, the service isn’t RESTful.
- The service is not resource-oriented if the scoping information isn’t in the URI.

Key principles

- Everything is a resource
- Each resource is identifiable by a unique identifier (URI)
- Use the standard HTTP methods
- Resources can have multiple representations
- Communicate statelessly

REST Constraints

- For a web service to be RESTful, it has to adhere to 6 constraints:
  - ✓ Client - Server separation
  - ✓ Stateless
  - ✓ Cacheable
  - ✓ Uniform interface
  - ✓ Layered system
  - ✓ Code-on-demand (Optional)

Client - Server separation

- The client and the server act independently, each on its own
  - ✓ Interaction between them is only in the form of
    - requests initiated by the client only
    - responses, which the server send to the client only as a reaction to a request

- The server sits there waiting for requests from the client to come
  - ✓ doesn't start sending away information about the state of some resources on its own
  - ✓ Responds only when a request comes in

Stateless

- Stateless means the server does not remember anything about the user who uses the service
  - ✓ doesn’t remember if the user already sent a GET request for the same resource in the past
  - ✓ doesn’t remember which resources the user of the API requested before
  - ✓ and so on...

- Each individual request contains all the information the server needs to perform the request and return a response, regardless of other requests made by the same user.
- The client is responsible for sending any state information to the server whenever it's needed.
- No session stickiness or session affinity on the server for the calling request

Cacheable

- Data the server sends contain information about whether or not the data is cacheable

- If the data is cacheable, it might contain some version number
  - ✓ version number is what makes caching possible
- The client knows which version of the data it already has (from a previous response)
  - ✓ the client can avoid requesting the same data again and again
- client should also know if the current version of the data is expired,
  - ✓ will know it should send another request to the server to get the most updated data about the state of a resource

Uniform interface

- There are four guiding principles suggested by Fielding that constitute the necessary constraints to satisfy the uniform interface
  - Identification of resources
  - Manipulation of resources
  - Self-descriptive messages
  - Hypermedia as the engine of application state
- The request to the server has to include a resource identifier
- Each request to the web service contains all the information the server needs to perform the request
  - ✓ Each response the server returns contain all the information the client needs to understand the response
- The response the server returns include enough information so the client can manipulate the resource
- Hypermedia as the engine of application state
  - ✓ Application mean the web application that the server is running
  - ✓ Hypermedia mean the hyperlinks, or simply links, that the server can include in the response
  - ✓ means that the server can inform the client , in a response, of the ways to change the state of the web application

HATEOS

- Hypermedia as the Engine of Application State

Without HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “owner” : “jim”
}
```

With HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “modify”,
                “href” : “/items/1234”
            },
            {
                “rel” : “delete”,
                “href” : “/items/1234”
            }
        ]
    }
}
```

Layered system

- Between the client who requests a representation of a resource's state, and the server who sends the response back
  - ✓ there might be a number of servers in the middle
  - ✓ servers might provide a security layer, a caching layer, a load-balancing layer etc.,
  - ✓ layers should not affect the request or the response

- The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request.

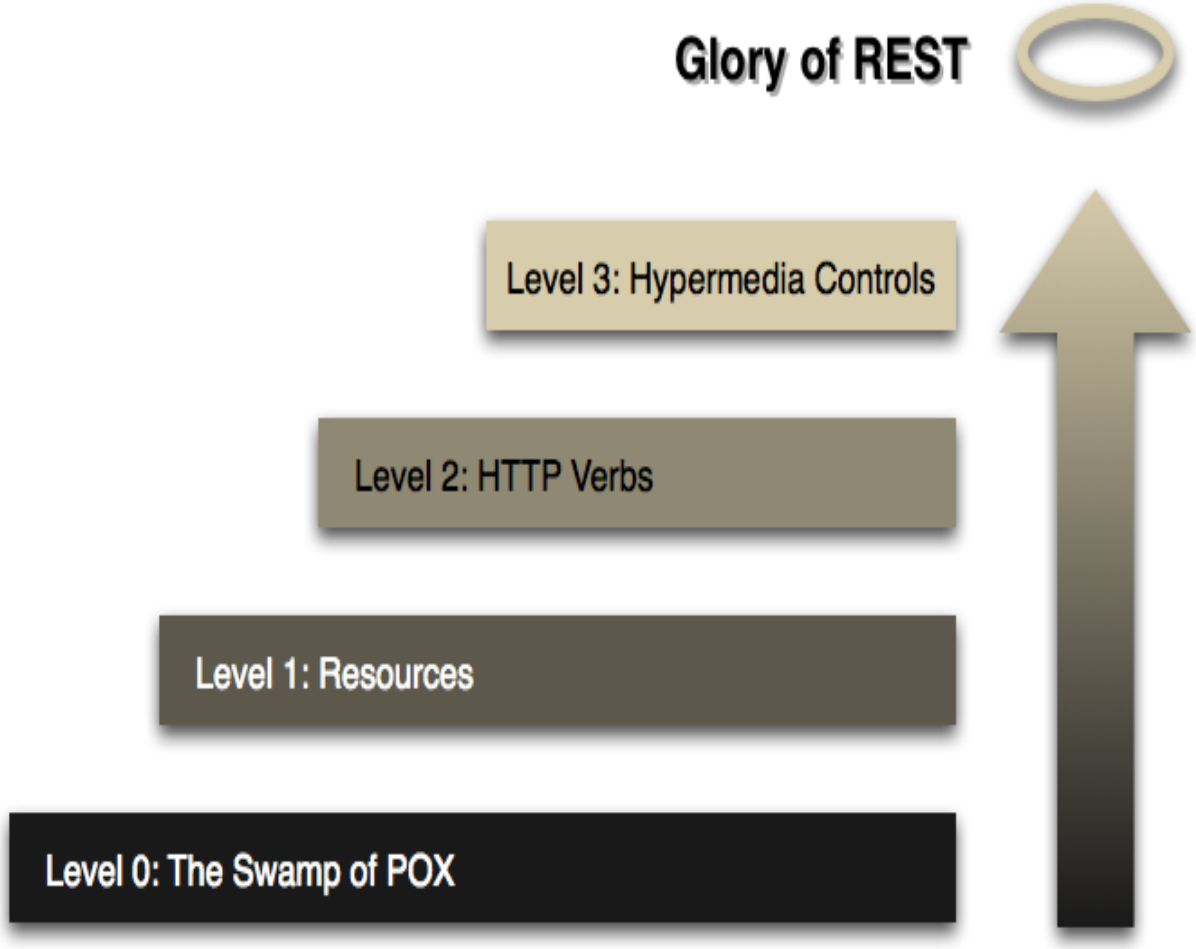
Code-on-demand (Optional)

- Is optional — a web service can be RESTful even without providing code on demand

- The client can request code from the server
  - ✓ response from the server will contain some code
  - ✓ when the response is in HTML format, usually in the form of a script
- The client then can execute that code

REST Maturity Model

- Richardson used three main factors to decide the maturity of service. These factors are
- URI,
- HTTP Methods,
- HATEOAS (Hypermedia)



REST API DESIGN

REST API Design

- API design is more of an art.
- Some best practices for REST API design are implicit in the HTTP standard.
  - When should URI path segments be named with plural nouns?
  - Which request method should be used to update the resource state?
  - How do I map *non-CRUD* operations to my URIs?
  - What is the appropriate HTTP response status code for a given scenario?
  - How can I manage the versions of a resource's state representations?
  - How should I structure a hyperlink in JSON?

API Design

- REST APIs are designed around *resources* , which is any kind of object, data, or service that is accessible to the client.
- A resource has an *identifier* , a URI that uniquely identifies that resource.
- The resource URIs should be based on nouns (the resource) and not verbs (the operations on the resource).

API Design Elements

- The following aspects of API design are all important, and together they define your API:
  - The representations of your resources and the links to related resources.
  - The use of standard (and occasionally custom) HTTP headers.
  - The URLs and URI templates define your API's query interface for locating resources based on their data.
  - Required behaviors by clients—for example, DNS caching behaviors, retry behaviors

- Interaction Design with HTTP(response codes, request methods)
- Identifier Design with URIs
- Metadata Design(Media Types, content negotiation)
- Representation Design(Message body format, Hypermedia Representation)

Interaction Design

- Interaction Design with HTTP
- REST APIs embrace all aspects of the HyperText Transfer Protocol including its request methods, response codes, and message headers
- Each HTTP method has specific, well-defined semantics within the context of a REST API’s resource model.
- The purpose of GET is to retrieve a representation of a resource’s state.
- HEAD is used to retrieve the metadata associated with the resource’s state.
- PUT should be used to add a new resource to a store or update a resource.
- DELETE removes a resource from its parent.
- POST should be used to create a new resource within a collection and execute controllers.

Interaction Design

- Interaction Design with HTTP

- Rule: GET and POST must not be used to tunnel other request methods
- Rule: GET must be used to retrieve a representation of a resource
- Rule: HEAD should be used to retrieve response headers
- Rule: PUT must be used to both insert a stored resource
- Rule: POST must be used to create a new resource in a collection
- Rule: POST must be used to execute controllers
- Rule: DELETE must be used to remove a resource from its parent
- Rule: OPTIONS should be used to retrieve metadata that describes a resource's available interactions

Interaction Design

HTTP response success code summary

- 200 OK Indicates a nonspecific success
- 201 Created Sent primarily by collections and stores but sometimes also by controllers to indicate that a new resource has been created
- 204 No Content Indicates that the body has been intentionally left blank
- 301 Moved Permanently Indicates that a new *permanent* URI has been assigned to the client's requested resource
- 303 See Other Sent by controllers to return results that it considers optional
- 401 Unauthorized Sent when the client either provided invalid credentials or forgot to send them
- 402 Forbidden Sent to deny access to a protected resource
- 404 Not Found Sent when the client tried to interact with a URI that the REST API could not map to a resource
- 405 Method Not Allowed Sent when the client tried to interact using an unsupported HTTP method
- 406 Not Acceptable Sent when the client tried to request data in an unsupported media type format
- 500 Internal Server Error Tells the client that the API is having problems of its own

Identifier Design

- REST APIs use Uniform Resource Identifiers (URIs) to address resources
- URI Format
- URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

- Forward slash separator (/) must be used to indicate a hierarchical relationship.
- Consistent subdomain names should be used for your APIs

Identifier Design

Resource Modelling

The URI path conveys a REST API's resource model,

Each forward slash-separated path segment corresponds to a unique resource within the model's hierarchy.

http://api.library.restapi.org/books/harry-potter

http://api.library.restapi.org/books

http://api.library.restapi.org/sections/fiction

http://api.library.restapi.org/sections

Identifier Design

document,

http://api.library.restapi.org/books/Pride and Prejudice

http://api.library.restapi.org/books/harry-potter/harry potter and the philosopher's stone

collection,

Each URI below identifies a collection resource:

http://api.library.restapi.org/books

http://api.library.restapi.org/sections

controller.

POST /alerts/245743/resend

Few antipatterns

GET /deleteUser?id=1234

GET /deleteUser/1234

DELETE /deleteUser/1234

POST /users/1234/delete

Identifier Design

URI Path Design

Rules

- A singular noun should be used for document names
- A plural noun should be used for collection names
- A verb or verb phrase should be used for controller names
- Variable path segments may be substituted with identity-based values http://api.library.restapi.org/section/{sectionId}/books/{bookId}/version/{versiond}
- CRUD function names should not be used in URIs

Identifier Design

- URI Query Design
- As a component of a URI, the query contributes to the unique identification of a resource.
- Rule: The query component of a URI may be used to filter collections
- Rule: The query component of a URI should be used to paginate collections
- **Represent relationships clearly in the URI**

Filter and Paginate

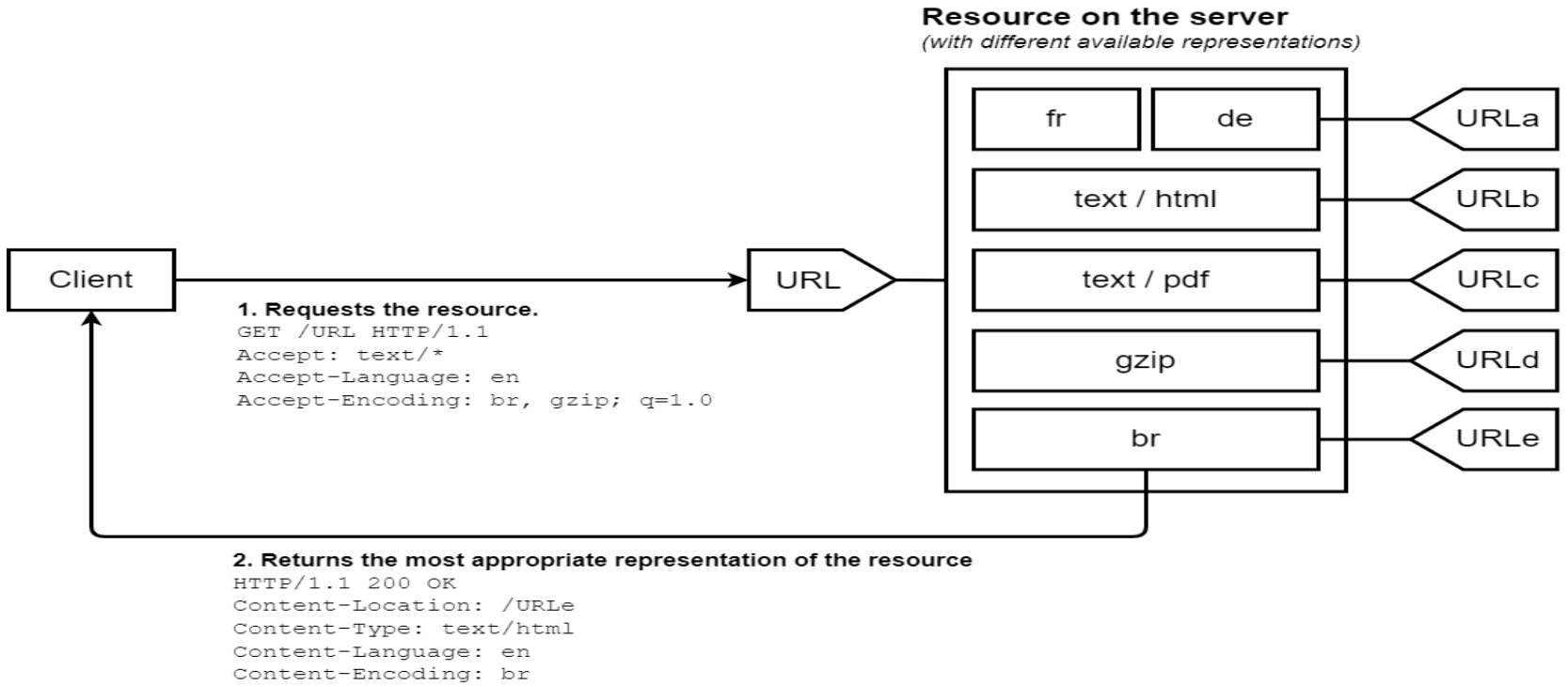
- */orders?minCost=n*
- /orders?limit=25&offset=50

Metadata Design

- Important HTTP headers
  - Content type
  - Content length
  - Last-modified
  - E-tag
  - Location

Metadata Design

- Media type negotiation should be supported when multiple representations are available



Representation Design

- Representation is the technical term for the data that is returned
- Users can view the representation of a resource in different formats, called media types
- JSON is the preferred format
- XML and other formats may optionally be used for resource representation
- A consistent form should be used to represent media types, links, and errors

Representation Design

- Keep the choice of media types and formats flexible to allow for varying application use cases, and client needs for each resource.
- Prefer to use well-known media types for representations.
- What data to include in JSON-formatted representations
- An example of a representation of a person resource:

```
{
"name" : "John",
"id" : "urn:example:user:1234",
"link" : {
"rel : "self",
"href" : "http://www.example.org/person/john"
},
"address" : {
"id" : "urn:example:address:4567",
"link" : {
"rel : "self",
"href" : "http://www.example.org/person/john/address"
}
```

Best Practices - Summary

- Use only nouns for a URI;
- GET methods should not alter the state of resource;
- Use plural nouns for a URI;
- Use sub-resources for relationships between resources;
- Use HTTP headers to specify input/output format;
- Provide users with filtering and paging for collections;
- Version the API;
- Provide proper HTTP status codes.

Best Practices

- Consistent API s - reduces the cognitive load on developers.
- Meaningful errors

  Example:

  Authentication failed because token is revoked : token_revoked or invalid_auth

  Value passed for name exceeded max: length name_too_long or invalid_name

API Design

- "*The API should enable developers to do one thing really well.It's not as easy as it sounds, and you want to be clear on what the API is not going to do as well.*"
    - *- Ido Green, developer advocate at Google*

RESTful Services Example - Customer

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

References

- RESTful Web APIs by Leonard Richardson and Mike Amundsen , Oreilly Media, 2013
- Web Development with Node and Express by Ethan Brown , Oreilly Media 2nd Edition , 2019

innovate achieve lead

BITS Pilani

Pilani Campus

Thank You!