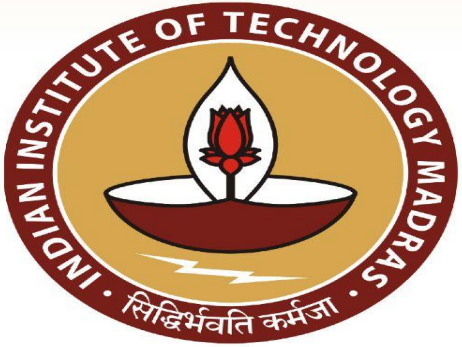


IIT Madras
BSc Degree

REST and APIs

API Design

- Web architecture - REST
- API Examples
- OpenAPI specification

Distributed Software Architecture

- Servers - Clients
- Standard “protocols” needed for communication
- Assumptions?
  - Server always on?
  - Server knows what client is doing?
  - Client authentication?
  - Network latency?

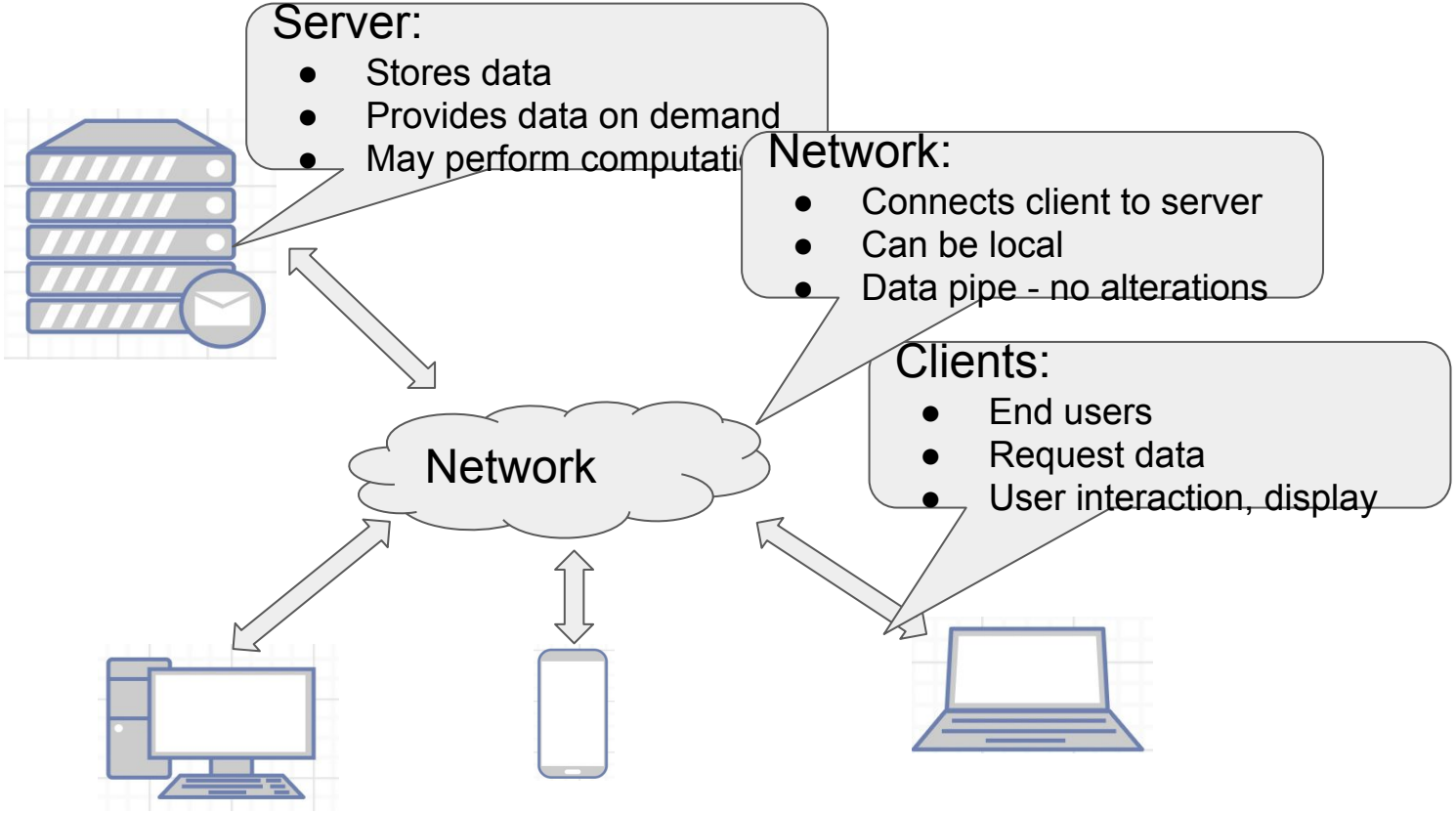
The Web

- Client - Server may be far apart
- Different networks, latencies, quality
- Authentication? Not core part of protocol
- State?
  - Server does not know what state client is in
  - Client cannot be sure what state server is in

Architecture for the Web

- Roy Fielding, PhD thesis 2000 UC Irvine
- “REpresentational State Transfer” - REST
  - Take into account limitations of the Web
  - Provide guidelines or constraints
- Software Architecture Style
  - Not a set of rules!

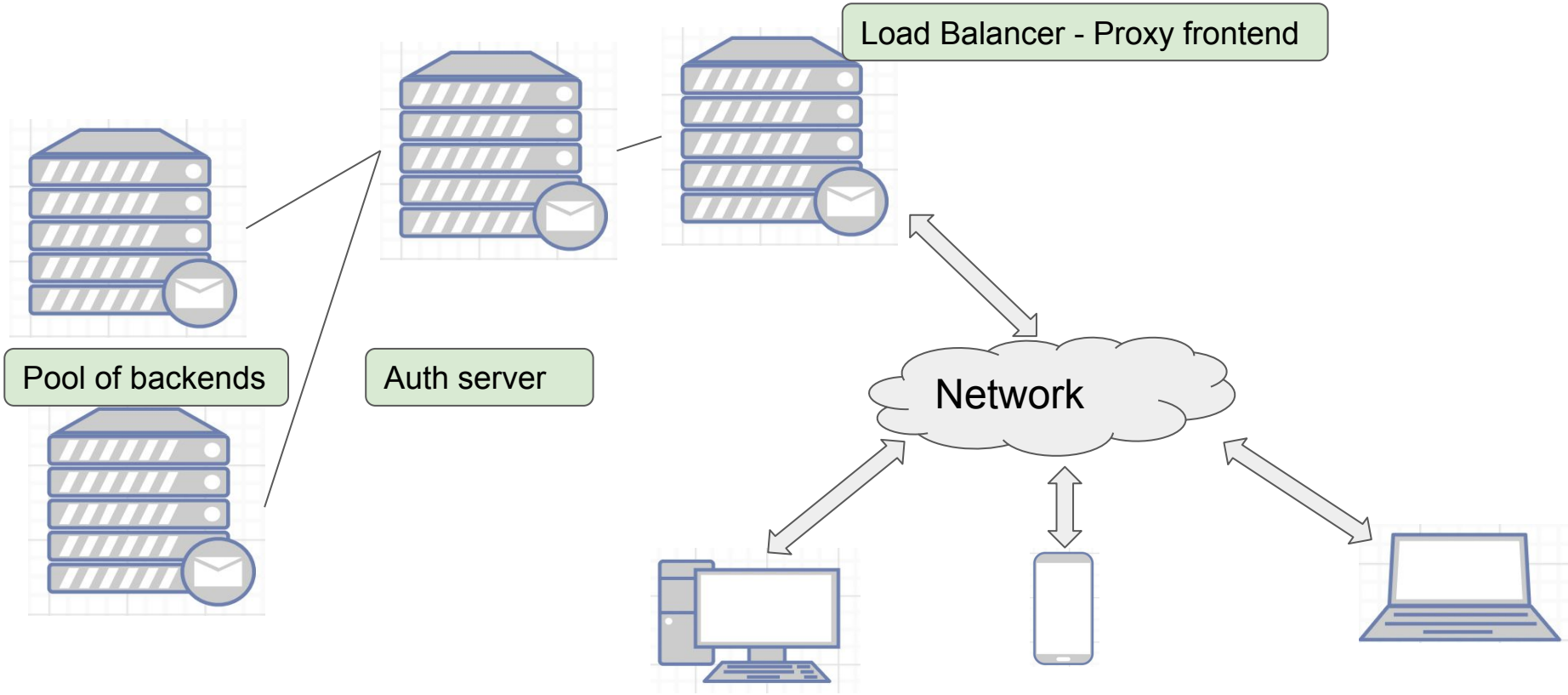
Constraint 1: Client - Server



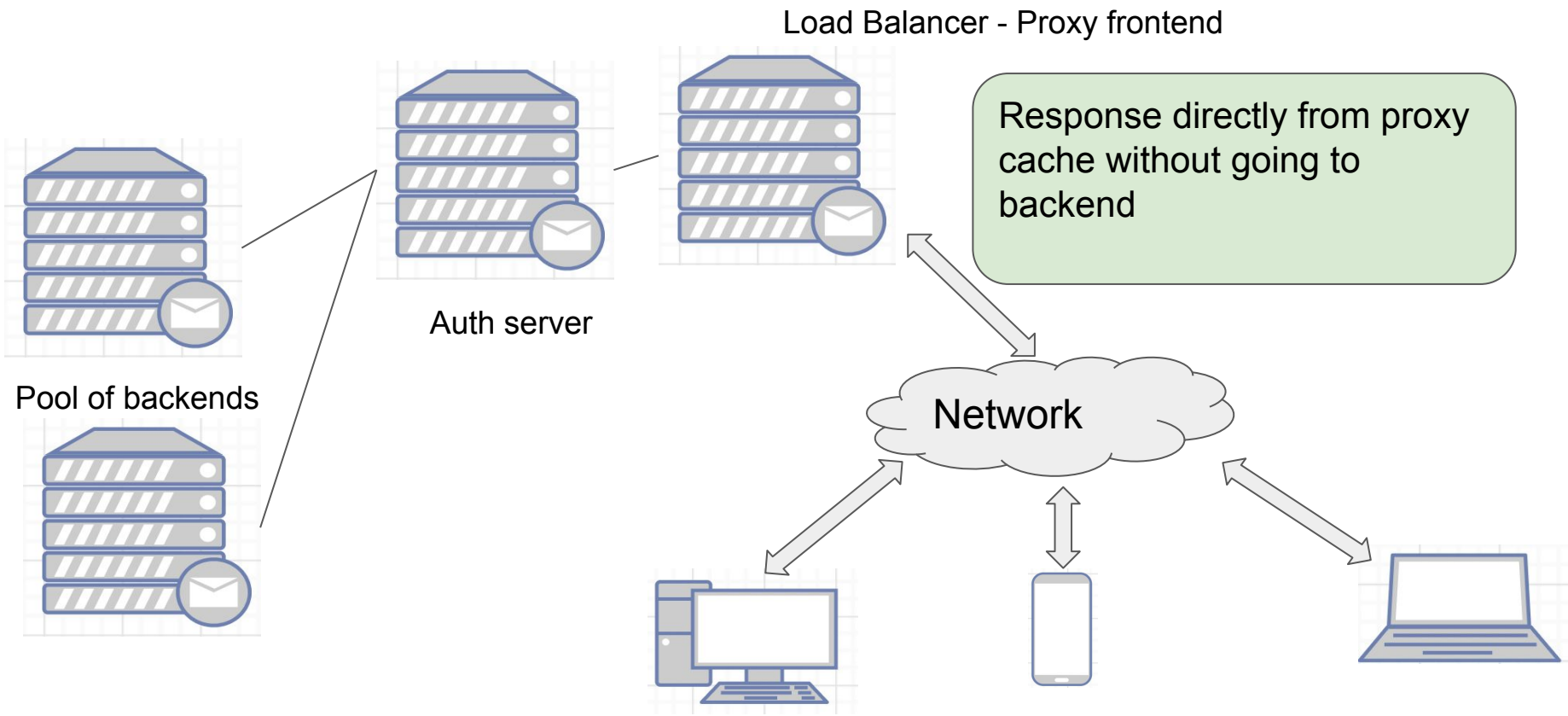
Constraint 2: Stateless

- Server cannot assume state of client:
  - which page are you looking at
  - is a request coming from an already logged in user just because of the address?
- Client cannot assume state of server:
  - did server reboot since last request?
  - is this request being answered by same server?

Constraint 3: Layered System



Constraint 4: Cacheability



Constraint 5: Uniform Interface

- Client and Server interact in a uniform and predictable manner
- Server exposes “resources”

Hypertext/media used to convey the available resources and functionality - can be discovered by client through hypertext information from server

(Optional) Constraint 6: Code on Demand

- Server can extend client functionality
  - Javascript
  - Java applets

Part of the overall architecture - these are not hard rules

REST

- REpresentational State Transfer

REST

- REpresentational State Transfer

What does that mean?

- State information between client and server explicitly transferred with every communication

Sequence

- Client accesses a Resource Identifier from server
  - Usually URI - superset of URL
  - Typically start from home page of application
  - No initial state assumed

Sequence

- Client accesses a Resource Identifier from server
  - Usually URI - superset of URL
  - Typically start from home page of application
  - No initial state assumed
- Resource Operation specified as part of access
  - If HTTP, then GET, POST etc.
  - Not fundamentally tied to protocol

Sequence

- Client accesses a Resource Identifier from server
  - Usually URI - superset of URL
  - Typically start from home page of application
  - No initial state assumed
- Resource Operation specified as part of access
  - If HTTP, then GET, POST etc.
  - Not fundamentally tied to protocol
- Server responds with new Resource Identifier
  - New state of system; new links to follow etc.

Sequence

- Client accesses a Resource Identifier from server
  - Usually URI - superset of URL
  - Typically start from home page of application
  - No initial state assumed
- Resource Operation specified as part of access
  - If HTTP, then GET, POST etc.
  - Not fundamentally tied to protocol
- Server responds with new Resource Identifier
  - New state of system; new links to follow etc.

State of interaction transferred back and forth

HTTP

- One possible protocol to carry REST messages
- Use the HTTP verbs to indicate actions
- Standardize some types of functionality

HTTP

- GET: Retrieve representation of target resource’s state
- POST: Enclose data in request: target resource “processes” it
- PUT: Create a target resource with data enclosed
- DELETE: Delete the target resource

Idempotent operations

- Repeated application of the operation is not a problem

Idempotent operations

- Repeated application of the operation is not a problem
- Example: GET is always safe - read-only operation

Idempotent operations

- Repeated application of the operation is not a problem
- Example: GET is always safe - read-only operation
- Example:
  - PUT: will always create the same new resource. If already exists, may give error

Idempotent operations

- Repeated application of the operation is not a problem
- Example: GET is always safe - read-only operation
- Example:
  - PUT: will always create the same new resource. If already exists, may give error
  - DELETE: can delete only once. may error on repeated deletion, but won’t change data

Idempotent operations

- Repeated application of the operation is not a problem
- Example: GET is always safe - read-only operation
- Example:
  - PUT: will always create the same new resource. If already exists, may give error
  - DELETE: can delete only once. may error on repeated deletion, but won’t change data
- POST: May NOT be idempotent
  - Example: Add comment to blog - repeat will cause multiple copies

CRUD

- CRUD - database operations
- Typically a common set of operations needed in most web applications
  - Good candidate for REST based functionality

REST != CRUD

But they do work well together

Data Encoding

- Basic HTML: for simple responses
- XML: Structured data response
- JSON: simpler form of structured data

Data serialization for transferring complex data types over text based format

JSON

- JavaScript Object Notation
- Nested arrays:
  - Serialize complex data structures like dictionaries, arrays etc.

JSON

- JavaScript Object Notation
- Nested arrays:
  - Serialize complex data structures like dictionaries, arrays etc.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

API data transfer format

- Input to API: text - HTTP
- Output: complex data types - JSON, XML, YAML etc.
  - JSON most commonly used
- Different from internal server representation
- Different from final view presentation

YAML

- Yet Another Markup Language - common alternative, especially for documentation and configuration

REST APIs

Some Examples

Typical functionality

- CRUD
- Variants of listing
- Specialized functions:
  - create a new virtual machine
  - reboot an existing virtual machine
  - Turn off street lights on a given street
- Formal specifications help others to use

Example: Wikipedia

- Open API
- Search for pages
- History of page
- JSON output

MediaWiki

API:REST API

Translate this page

Other languages: English • polski • 日本語

Main page
Get MediaWiki
Get extensions
Tech blog
Contribute

Support
User help
FAQ
Technical manual
Support desk
Communication

Development
Bug tracker

This page is part of the MediaWiki REST API documentation.

The MediaWiki Core REST API lets you interact with MediaWiki by sending HTTP requests to unique URLs. You can use the API to build apps and scripts that search and display wiki pages, get media files, and explore page history.

Quickstart

```
# Search English Wikipedia for an article about Earth
$ curl "https://en.wikipedia.org/w/rest.php/v1/search/page?
q=earth&limit=1"
```

```
curl "https://en.wikipedia.org/w/rest.php/v1/search/page?q=earth&limit=1"
```

```
curl "https://en.wikipedia.org/w/rest.php/v1/search/page?q=earth&limit=1"
```


```
{"pages":[
 {"id":9228,"key":"Earth","title":"Earth",
 "excerpt":"<span class=\"searchmatch\">Earth</span> is the third
planet from the Sun and the only astronomical object known to
harbor and support life. About 29.2% of <span
class=\"searchmatch\">Earth</span>'s surface is land consisting",
 "description":"Third planet from the Sun in the Solar System",
 "thumbnail":{"mimetype":"image/jpeg","size":null,"width":200,
"height":200,"duration":null,"url":"//upload.wikimedia.org/wikipe
dia/commons/thumb/9/97/The_Earth_seen_from_Apollo_17.jpg/200px-Th
e_Earth_seen_from_Apollo_17.jpg"}
  }
]}
```

`https://en.wikipedia.org/w/rest.php/v1/page/Main_Page/history`


```
{"revisions":[
{"id":1004593520,"timestamp":"2021-02-03T11:11:30Z","minor":
false,
  "size":3508,"comment":"cut",
  "user":{"id":2927383,"name":"Izno"},"delta":28},
{"id":1004592788,"timestamp":"2021-02-03T11:03:39Z","minor":
false,
"size":3480,"comment":"cut"
. . .
```

Documentation

Schema [edit]

`id` required	integer	Page identifier
`key` required	string	Page title in URL-friendly format
`title` required	string	Page title in reading-friendly format
`excerpt` required	string	*For search pages endpoint:* A few lines giving a sample of page content with search terms highlighted with `<span class=\"searchmatch\">` tags *For autocomplete page title endpoint:* Page title in reading-friendly format
`description` required	string	Short summary of the page topic based on the corresponding entry on Wikidata or `null` if no entry exists

Detailed documentation

Search pages [edit]

Route	`/search/page?q=search terms`	**Content type**	`application/json`
Method	`GET`	**Returns**	`pages` object containing array of search results

Searches wiki page titles and contents for the provided search terms, and returns matching pages.

When using this endpoint on your wiki
This endpoint uses the search engine configured in the $wgSearchType configuration setting and returns results in the namespaces configured by $wgNamespacesToBeSearchedDefault.

Examples [edit]

curl | Python | PHP | JavaScript

```
# Search English Wikipedia for up to 20 pages containing information about Jupiter
$ curl https://en.wikipedia.org/w/rest.php/v1/search/page?q=jupiter&limit=20
```

Parameters and Response Codes

Parameters [edit]

`q` required | query	Search terms
`limit` optional | query	Maximum number of search results to return, between 1 and 100. Default: 50

Responses [edit]

200	Success: Results found. Returns a `pages` object containing an array of search results.
200	Success: No results found. Returns a `pages` object containing an empty array.
400	Query parameter not set. Add `q` parameter.
400	Invalid limit requested. Set `limit` parameter to between 1 and 100.
500	Search error

Example: CoWin public APIs

- For Co-Win app: vaccine registration and information
- Unauthenticated APIs:
  - statewise search, districts etc.
- Authenticated APIs:
  - Book appointment

https://apisetu.gov.in/public/marketplace/api/cowin#/

General Information

Servers

https://cdn-api.co-vin.in/api - Production Server

Authorize

User Authentication APIs

Metadata APIs

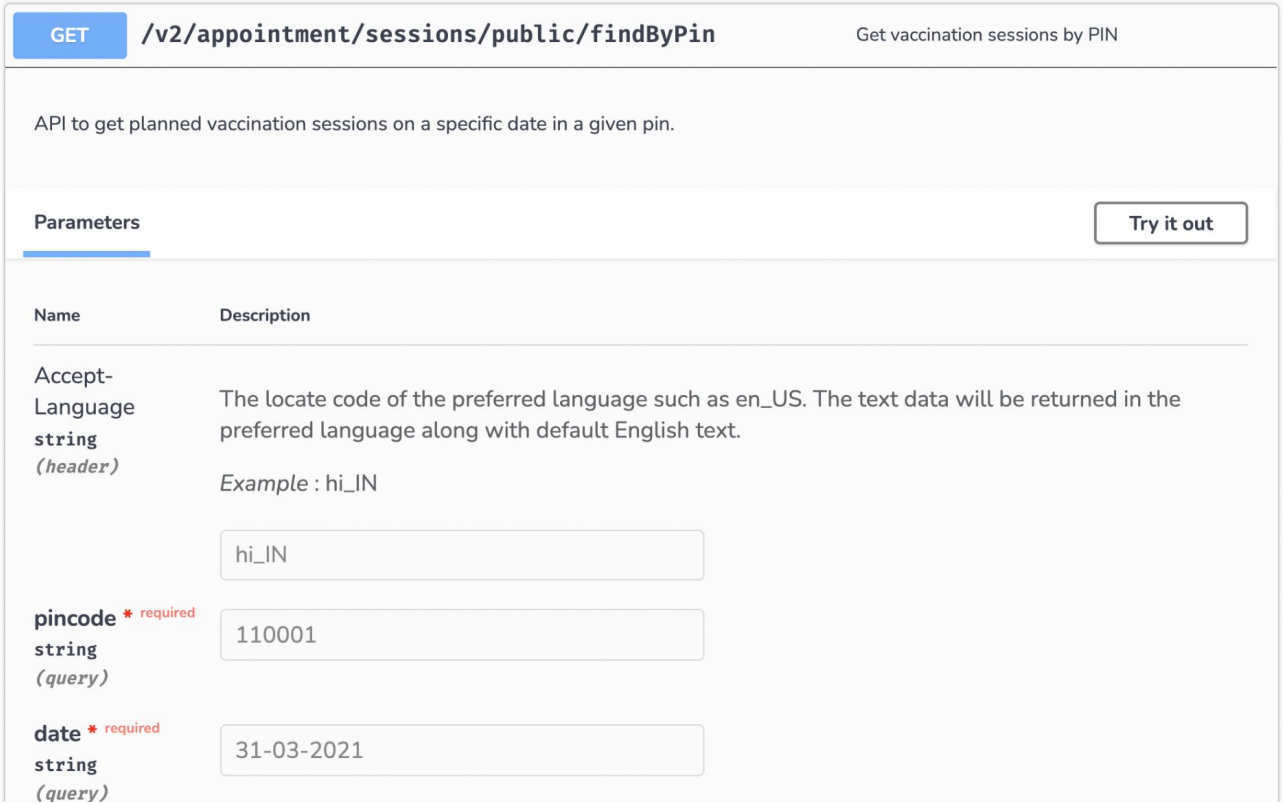
Appointment Availability APIs

Certificate APIs

Schemas

CentersSchema

Example: Availability API



Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/appointment/sessions/public/findByPin" -H
"accept: application/json" -H "Accept-Language: en_US"
```

Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/appointment/sessions/public/findByPin" -H
"accept: application/json" -H "Accept-Language: en_US"
```

```
{"errorCode":"USRRES0001","error":"Input parameter missing"}
```

Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/appointment/sessions/public/findByPin?pincod
e=600020&date=04-08-2021" -H "accept: application/json" -H "Accept-Language:
en_US"
```

Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/appointment/sessions/public/findByPin?pincod
e=600020&date=04-08-2021" -H "accept: application/json" -H "Accept-Language:
en_US"
```


```
      {"sessions":[
      {"center_id":604384,
      "name":"Fortis Malar Hospital",
      "address":"Fortis Malar HospitalChennai TN.",
      "state_name":"Tamil Nadu",
      "district_name":"Chennai",
      "block_name":"Adyar",
      "pincode":600020,
      "from":"13:30:00","to":"15:30:00","lat":12,"long":80,"fee_type":"Paid",
      "session_id":"d40bd2c9-0f42-4948-b794-e3c31fa7c3cc",
      "date":"04-08-2021",
      "available_capacity":85,
      "available_capacity_dose1":40,
      "available_capacity_dose2":45,
      "fee":"1250", . . .
```

Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/registration/certificate/public/download?ben
eficiary_reference_id=1234567890123" -H "accept: application/json" -H
"Accept-Language: en_US" -H "User-Agent: Mozilla/5.0"
```

Testing public API: WARNING - do not overdo this!

```
curl -X GET
"https://cdn-api.co-vin.in/api/v2/registration/certificate/public/download?ben
eficiary_reference_id=1234567890123" -H "accept: application/json" -H
"Accept-Language: en_US" -H "User-Agent: Mozilla/5.0"
```

Unauthenticated access!

Authentication

- Many APIs must be protected:
  - only meant for specific users
  - avoid abuse by overloading servers

How?

Authentication

- Many APIs must be protected:
  - only meant for specific users
  - avoid abuse by overloading servers

How?

Require a “token” that only a valid user can have

- Securely give token only when user logs in - Google OAuth, Facebook etc.
- API Key: one time token that user downloads - can be copied, so potentially less secure unless combined with other methods

Summary

- API examples: CoWin, Google cloud, Twitter, github, …
- Authentication may be enforced or optional on some parts
- Allows third-party integrations
- Equivalent of a “remote procedure call” - call a function on a remote system

OpenAPI

APIs of interest for web apps

- Purpose: information hiding - neither server nor client should know details of implementation on other side
- Unbreakable contract: should not change - standardized
  - Versions may update with breaking changes

Documentation?

- Highly subjective - some programmers better than others at documenting
- Incomplete - what one programmer finds important may not match others
- Outdated
- Human language specific

Description Files

- Machine readable - has very specific structure
- Enable automated processing:
  - boilerplate code
  - mock servers
- Example: assembly language is a version of the programming language of computers that is both machine and human readable
  - Structured so it can be compiled
- Versus: English language specification which needs someone to write code

OpenAPI Specification (OAS)

- **Vendor-neutral** format for **HTTP-based remote API** specification
- Does not aim to describe all possible APIs
- Efficiently describe the common use cases
- Originally developed as Swagger - evolved from Swagger 2.0

Current version: OAS3 - v3.1.0 as of Aug 2021

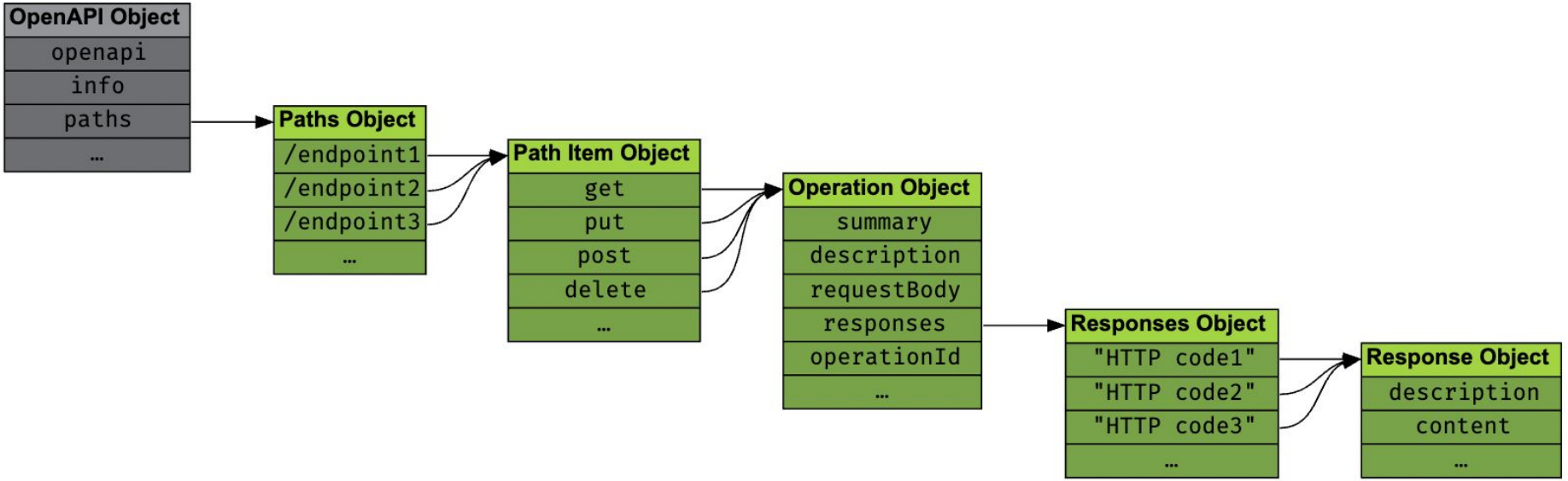
Concepts

- Describe in YAML (or possibly JSON)
- Specific structure to indicate overall information, paths, schemas etc.

eg:

```yaml
openapi: 3.1.0
info:
  title: A minimal OpenAPI document
  version: 0.0.1
paths: {} # No endpoints defined
```

Endpoints List



From: https://oai.github.io/Documentation/specification-paths.html

Paths

```yaml
openapi: 3.1.0
info:
  title: Tic Tac Toe
  description: |
    This API allows writing down marks on a Tic Tac Toe board
    and requesting the state of the board or of individual squares.
  version: 1.0.0
paths:
  /board:
    ...
```

Operations

```
paths:
  /board:
    get:
      ...
    put:
      ...
```

Operation object

```yaml
paths:
  /board:
    get:
      summary: Get the whole board
      description: Retrieves the current state of the board and the winner.
      parameters:
        ...
      responses:
        ...
```

Responses

```
paths:
  /board:
    get:
      responses:
        "200":
          ...
        "404":
          ...
```

Response Objects

```
paths:
  /board:
    get:
      responses:
        "200":
          description: Everything went fine.
          content:
            ...
```

Content Specification

```
content:
  application/json:
    ...
  text/html:
    ...
  text/*:
    ...
```

Schema

```
content:
  application/json:
    schema:
      type: integer
      minimum: 1
      maximum: 100
```

Complex Schema

```yaml
content:
  application/json:
    schema:
      type: object
      properties:
        productName:
          type: string
        productPrice:
          type: number
```

Parameters

```yaml
paths:
  /users/{id}:
    get:
      parameters:
      - name: id
        in: path
        required: true
```

Request Body

```
requestBody:
  content:
    application/json:
      schema:
        type: integer
        minimum: 1
        maximum: 100
```

Best Practices

- Design-first vs Code-first
  - Always prefer design-first!
- Single source of truth
  - The structure of the code should be *derived* from the OAS - *or* -
  - Spec should be derived from code
  - Minimize chances of code and documentation diverging
- Source code version control
- OpenAPI is … Open - public documentation better to identify problems
- Automated tools, editors - make use of them!