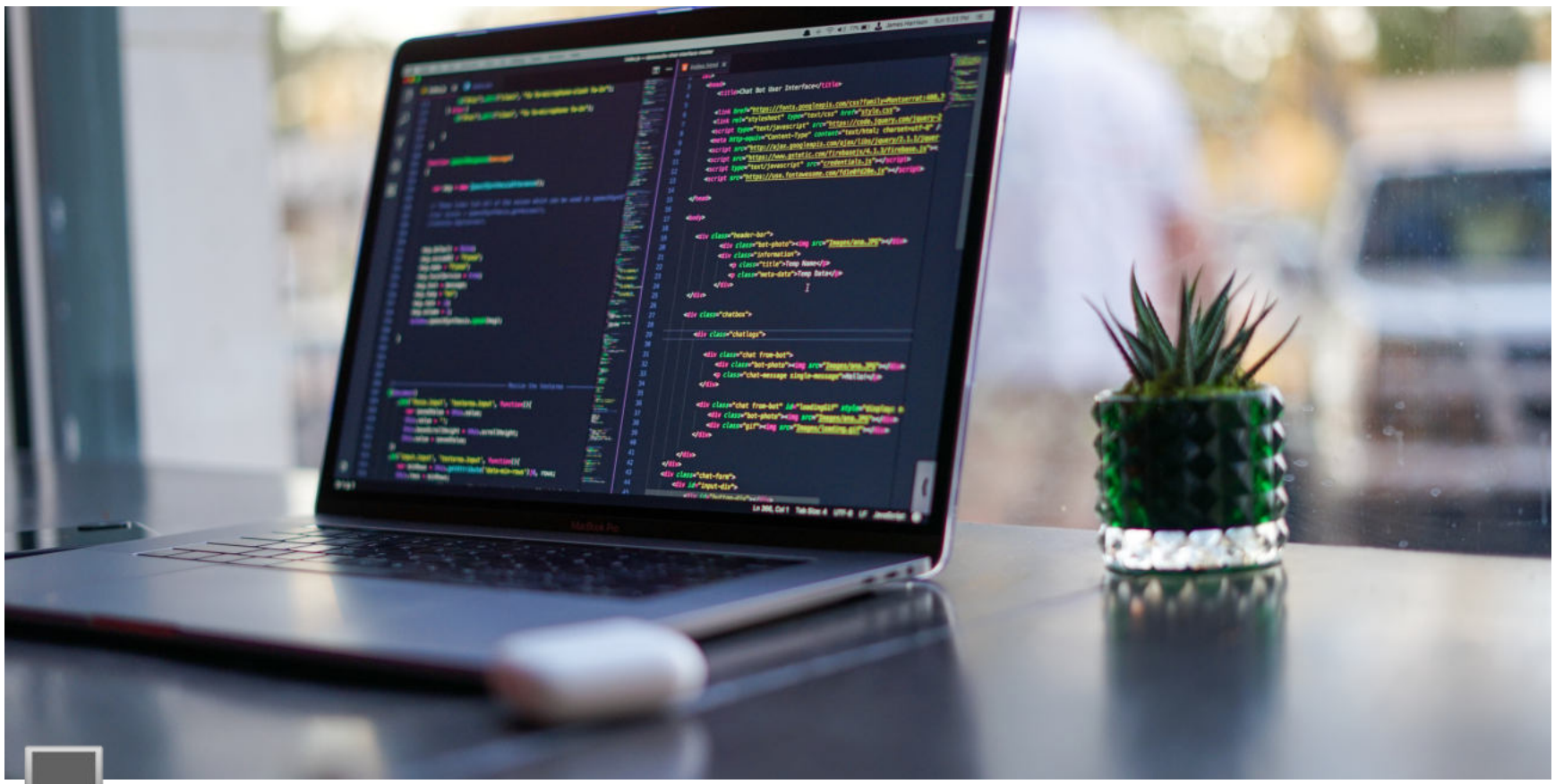


Week 6 Lecture 1

Class	BSCCS2003
Created	@October 11, 2021 12:36 PM
Materials	
Module #	32
Type	Lecture
Week #	6

API Design

Distributed Software Architecture

- Server - 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 network, 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

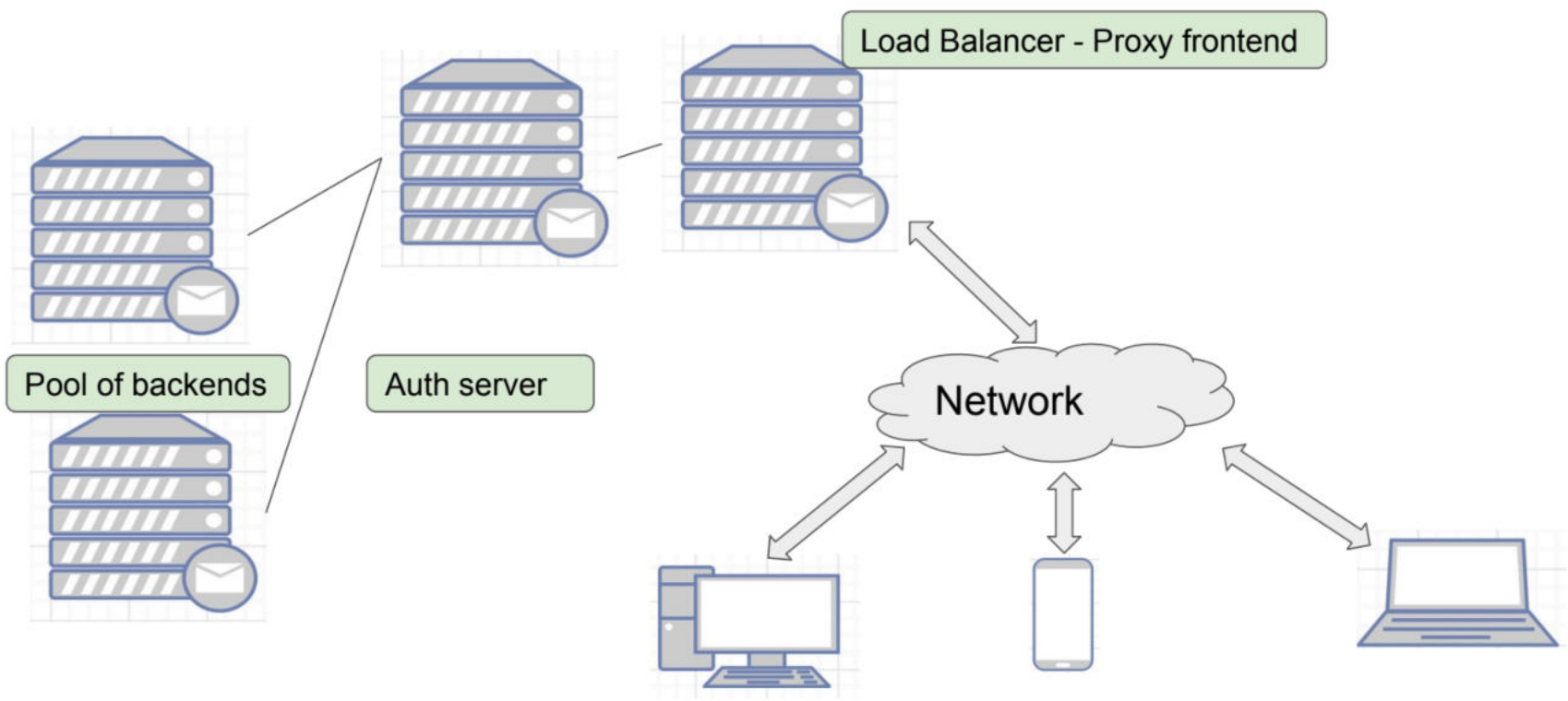
- "REpresentational State Transfer" - REST
  - Take into account the limitations of the web
  - Provide guidelines or constraints
- Software Architecture Style
  - Not a set of rules

Constraint #1: Client-Server

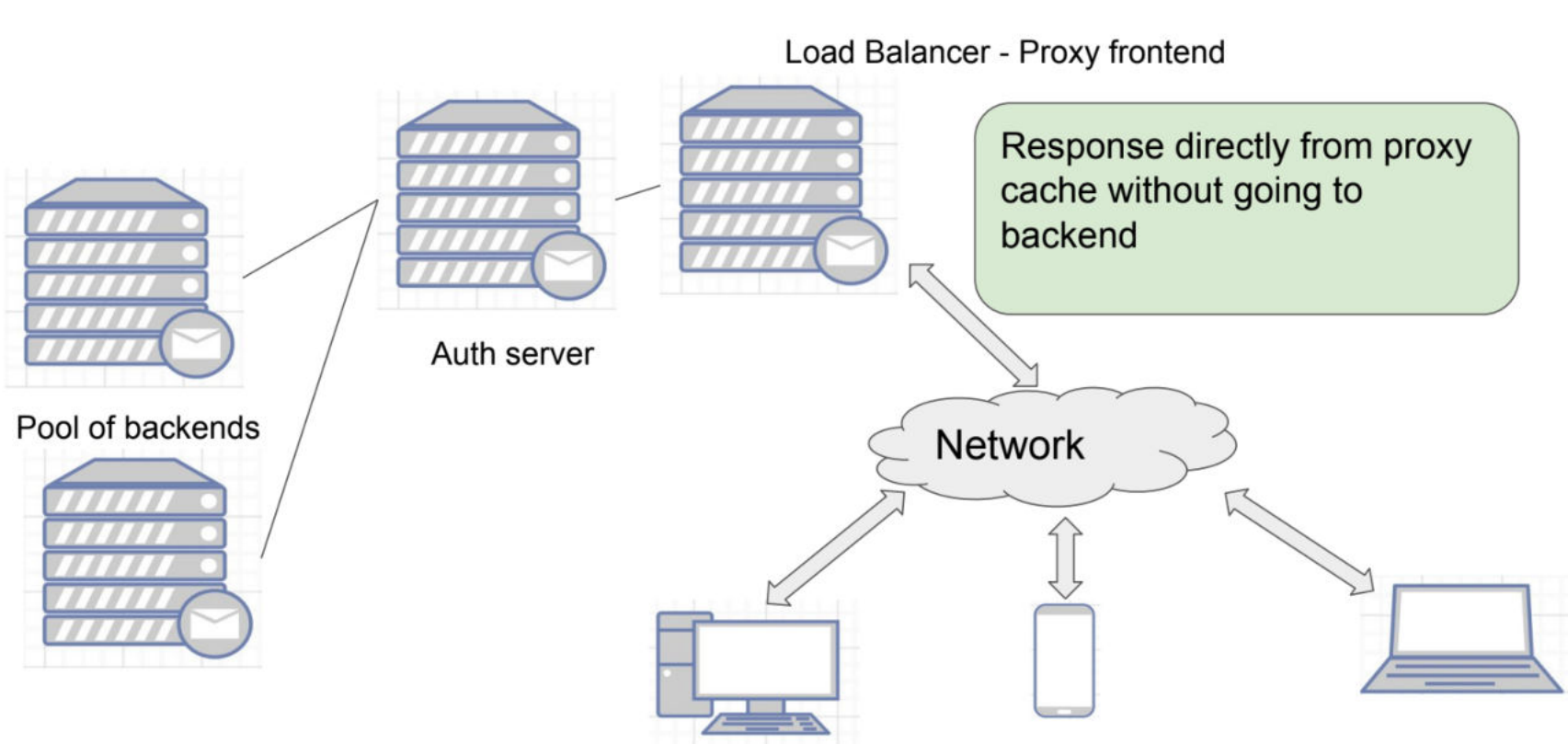
Constraint #2: Stateless

- Server cannot assume state of the 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 the state of the server
  - Did the server reboot since the last request?
  - Is this request being answered by the same server?

Constraint #3: Layered system



Constraint #4: Cacheability



Constraint #5: Uniform interface

- Client and State 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 the server

Constraint #6: Code on Demand (Optional)

- Server can extend client functionality
  - JavaScript
  - Java applets (are they still relevant?)

Part of the overall structure - these are not hard rules

Week 6 Lecture 2

Class	BSCCS2003
Created	@October 11, 2021 7:15 PM
Materials	
Module #	33
Type	Lecture
Week #	6

REST

REST

- REpresentational State Transfer

What does that mean?

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

Sequence

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

- Use the HTTP verbs to indicate action
- Standardize some types of functionality

- **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
- **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 the data
  - **POST:** May NOT be idempotent
    - **Example:** Add comment to a blog - repeat will cause multiple copies

CRUD

- CRUD: Database operations
- Create, Read, Update, Delete
- Typically a common set of operations needed in most web applications
  - Good candidate of REST based functionality

REST $\neq$ CRUD

But, they do work together quite well

Data Encoding

- **Basic HTML:** for simple reasons
- **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.

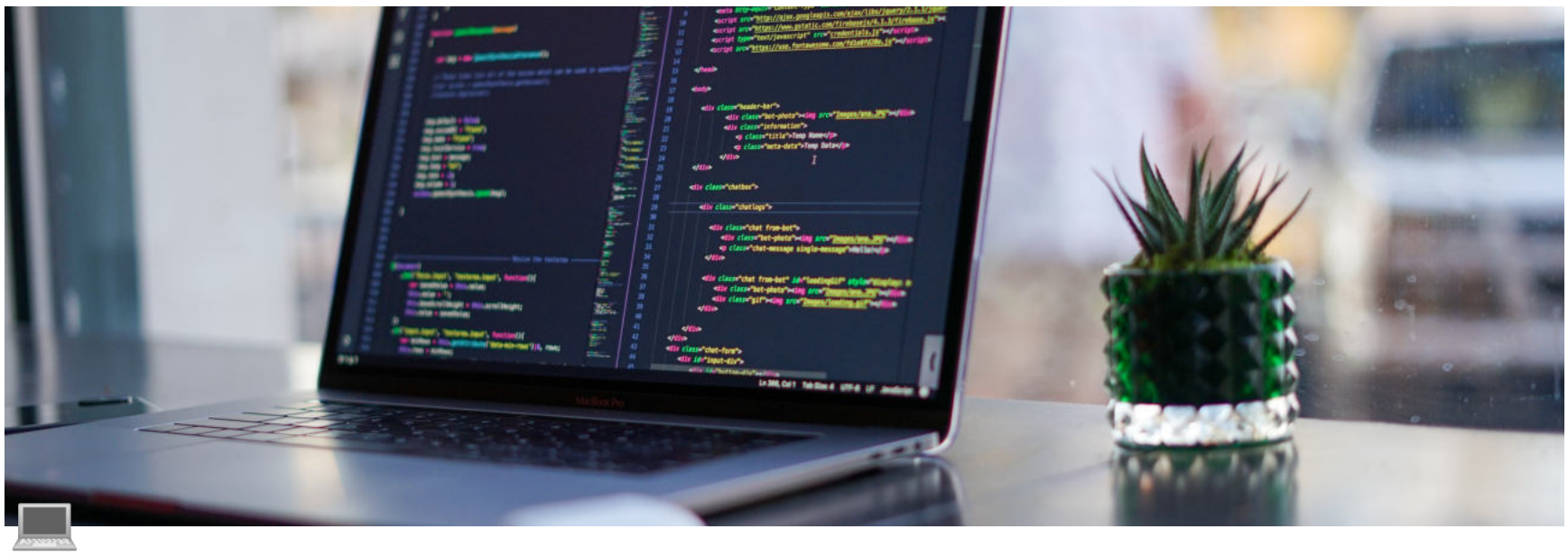
```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"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



Week 6 Lecture 3

Class	BSCCS2003
Created	@October 11, 2021 7:17 PM
Materials	
Module #	34
Type	Lecture
Week #	6

REST APIs

- 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 #1: Wikipedia

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

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

Response


```
{
  "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 harbour and support life. About 29.2% of <span class=\"searc
      "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/wikipedia/commons/thumb/9/97/The_Earth_seen_from_Apollo_17.jpg/200px-The_Earth_seen_from_Apollo_17.jpg"
      }
    }
  ]
}
```


Documentation

Schema [edit]

`id` required I integer	Page identifier
`key` required I string	Page title in URL-friendly format
`title` required I string	Page title in reading-friendly format
`excerpt` required I 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 I 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
```

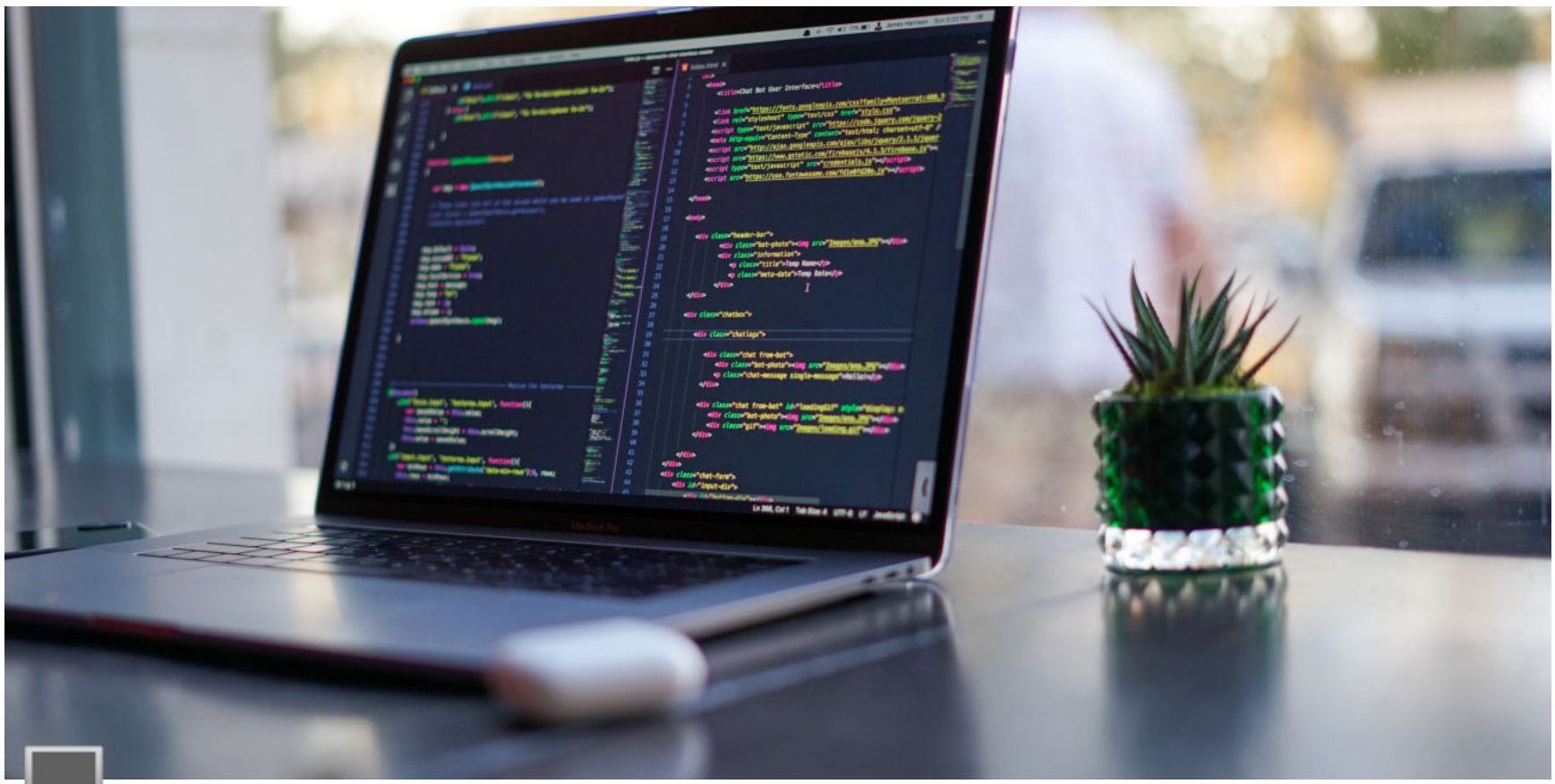
Parameters and Response Codes

Parameters [edit]

`q` required I query	Search terms
`limit` optional I 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



Week 6 Lecture 4

Class	BSCCS2003
Created	@October 11, 2021 8:07 PM
Materials	
Module #	35
Type	Lecture
Week #	6

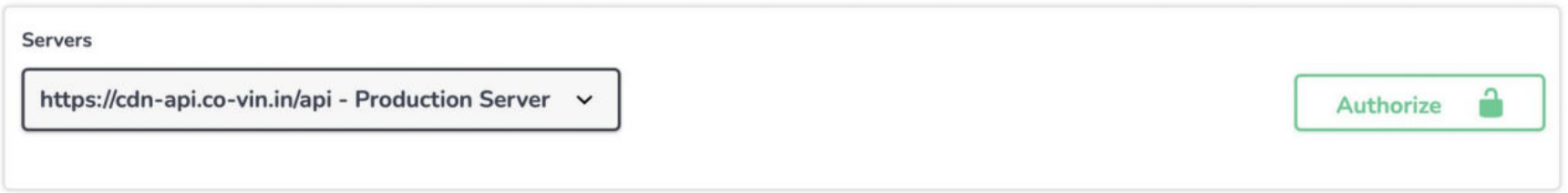
REST APIs

Example #2: CoWin public APIs

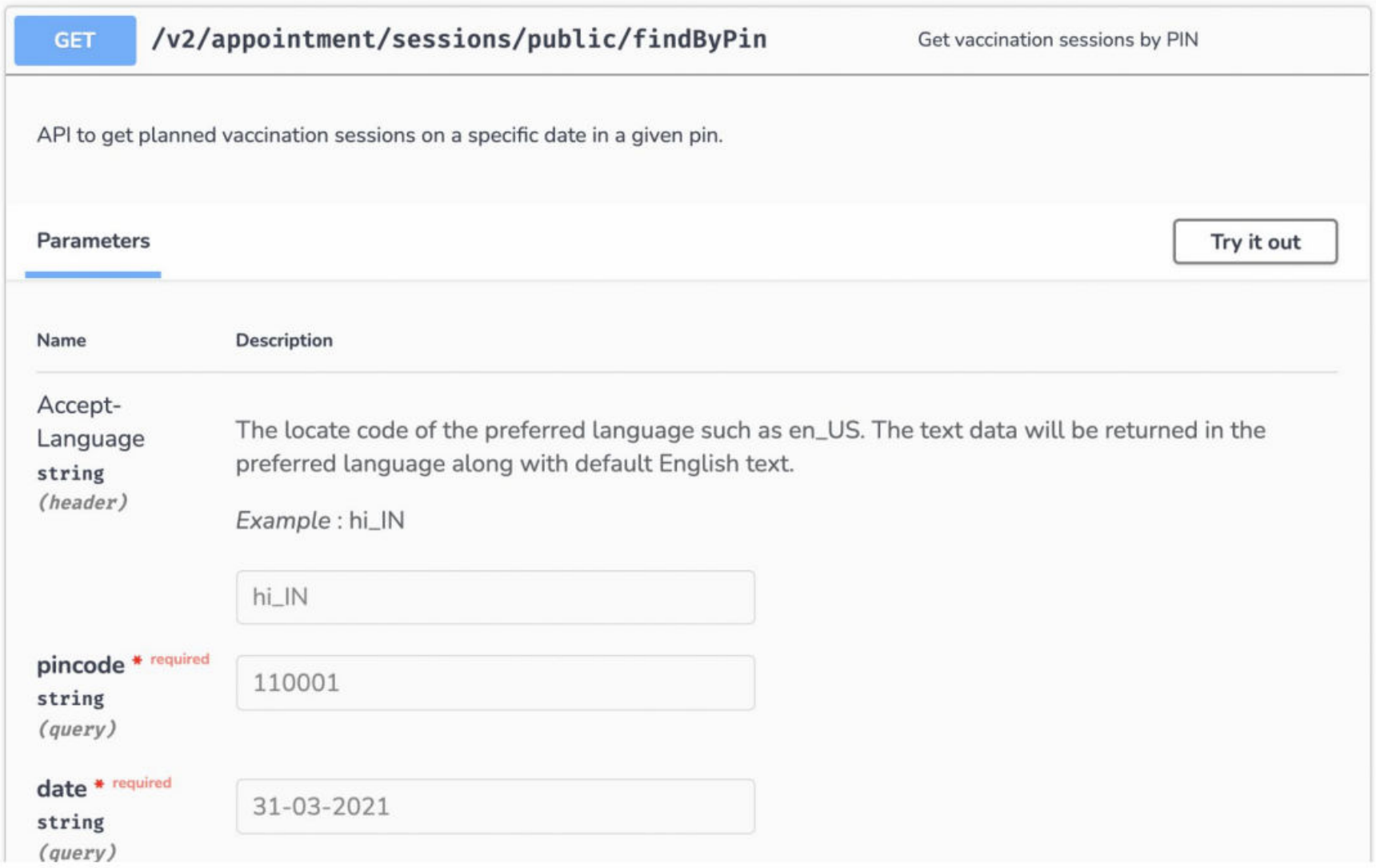
- For Co-Win app: Vaccine registration and information
- Unauthenticated APIs:
  - State-wise search, districts, etc.
- Authenticated APIs:
  - Book appointment

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

General Information



Example: Availability API



Testing public API

WARNING: Do NOT overdo this (This is a public API related to health services, overloading this might have some serious implications especially on someone's life)

```
$ 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"
}
```

```
$ curl -X GET "https://cdn-api.co-vin.in/api/v2/appointment/sessions/public/findByPin?pincode=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", . . .
```


```
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 to implement this?

Require a "token" that only a valid user can have

- Securely give token only when the 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,... etc.
- Authentication may be enforced or optional on some parts
- Allows 3rd party integrations
- Equivalent of a "remote procedure call" — call a function on a remote system

Week 6 Lecture 5

Class	BSCCS2003
Created	@October 11, 2021 8:32 PM
Materials	
Module #	36
Type	Lecture
Week #	6

OpenAPI

APIs of interest for web apps

- **Purpose:** information hiding — neither server nor the client should know the details of the implementation on the 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 that 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

Week 6 Lecture 6

Class	BSCCS2003
Created	@October 11, 2021 8:49 PM
Materials	
Module #	37
Type	Lecture
Week #	6

Important Concepts of an API

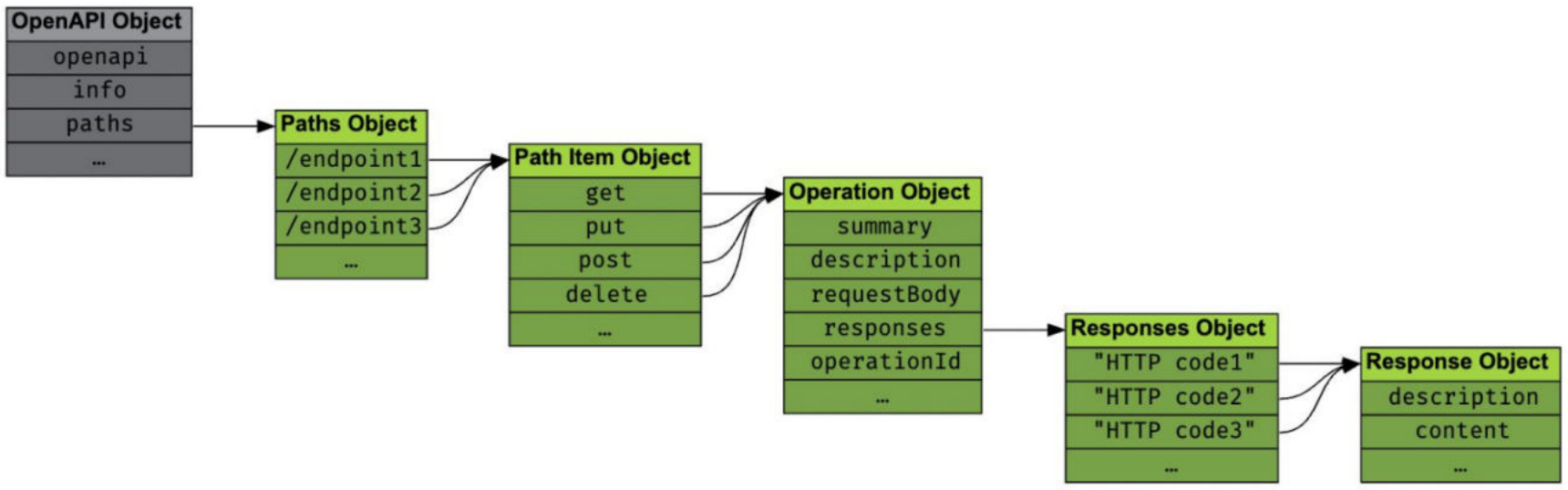
Concepts

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

eg:

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

Endpoints List



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

Paths

```
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

```
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

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

Parameters

```
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 v/s Code-first
  - Always prefer the design-first
- Single source of truth
  - The structure of the code should be derived from the OAS - or -
  - Spec should be derived from the code
  - Minimize chances of code and documentation diverging
- Source code version control
- OpenAPI is ... Open — public documentation better to identify the problems

- Automated tools, editors — make use of them