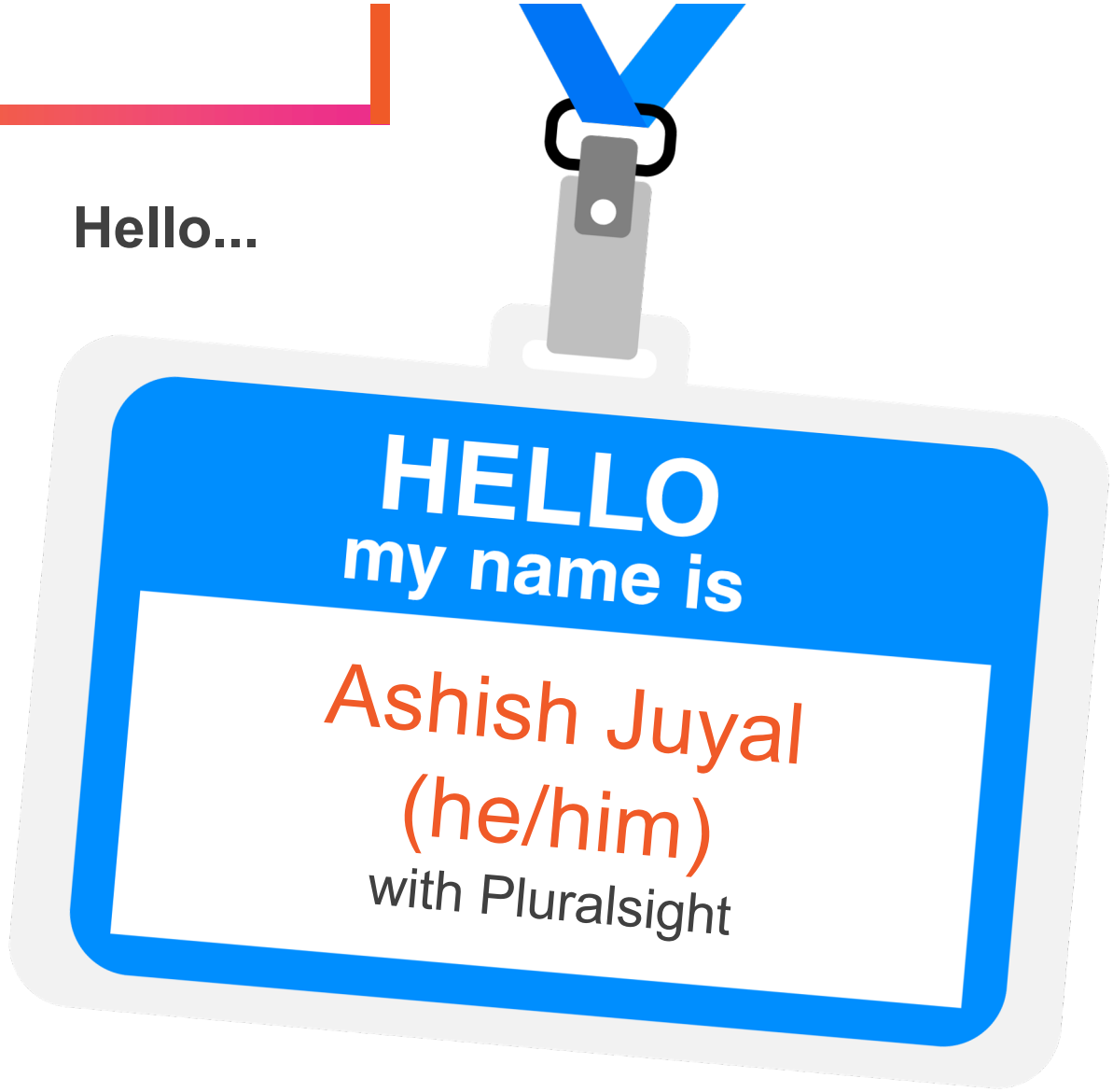
Welcome
Introduction to API Design and Development

DevelopIntelligence
A PLURALSIGHT COMPANY

Hello...



About me…

- 24+ years experience
- Architect & TDD Coach
- Consultancy
- Distributed architecture
- Microservices
- DevOps
- Java / Scala / Go

Prerequisites

This course assumes you:

- This course assumes you are familiar with the concepts of server-client relationship and the core needs of data delivery to/from servers/clients.
- This course will cover the common concepts of API Design and development focusing on RESTful JSON API design.

Why RESTful JSON?

- Widely used, supported API structure
- Flexible but defined
- Focused on data (not functionality)
- Can be a facade
- Human Readable
- Cross-platform, language, etc.

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

Objectives

At the end of this course you will be able to:

- Understand the design and structure of a RESTful API
- Be able to define a RESTful JSON spec (OpenAPI)
- Become familiar with tools related to API design/development
- Understand various concepts and approaches to API design
- Understand concepts related to API testing
- Determine a course of action for developing an API

Agenda

- Introduction
- JSON
- Key Aspects of REST
- API Design
- OpenAPI API Specification
- Design Concepts
- Richardson Maturity Model
- Testing and Development
- Development Standards
- Contracts
- Design First vs Code First
- Automated Testing
- Versioning
- Security
- API Design Details
- Automated Testing Details

How we’re going to work together

- Discussion
- Demo
- Do!

Breaks

10 mins break every hour

Housekeeping

📞 If I drop, I'll be back

Working with virtual machines

We will be doing most of our work in virtual machines

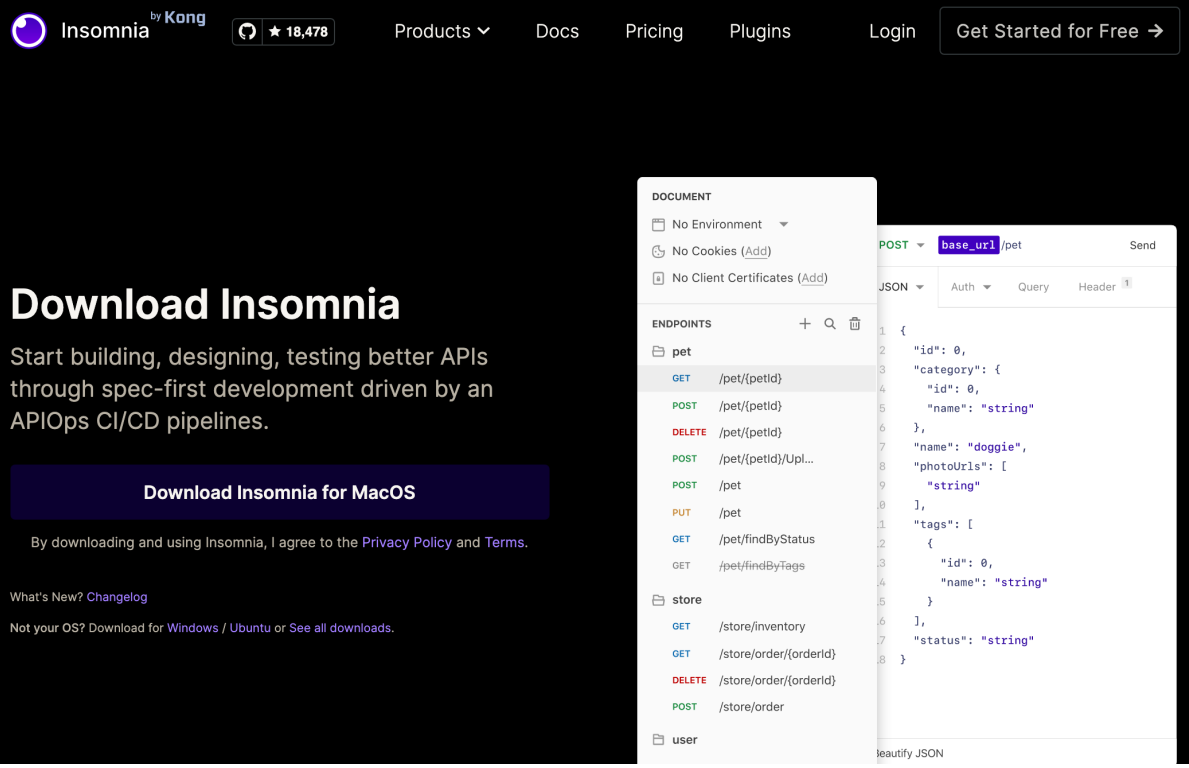
1. Please login to https://cloud.datacouch.io/pluralsight/ in an incognito window in Chrome
    - Just do File -> New Incognito Window
2. Note that to copy/paste to/from the VM you will need to use CTRL+SHIFT+C and CTRL+SHIFT+V on the mac
3. Note that Firefox, VSCode and other tools are already installed

Introduction to API Design and Development

Understanding APIs in Software Delivery: Introduction

Understanding APIs in Software Delivery: Introduction

- **Install Insomnia**
  https://insomnia.rest/download

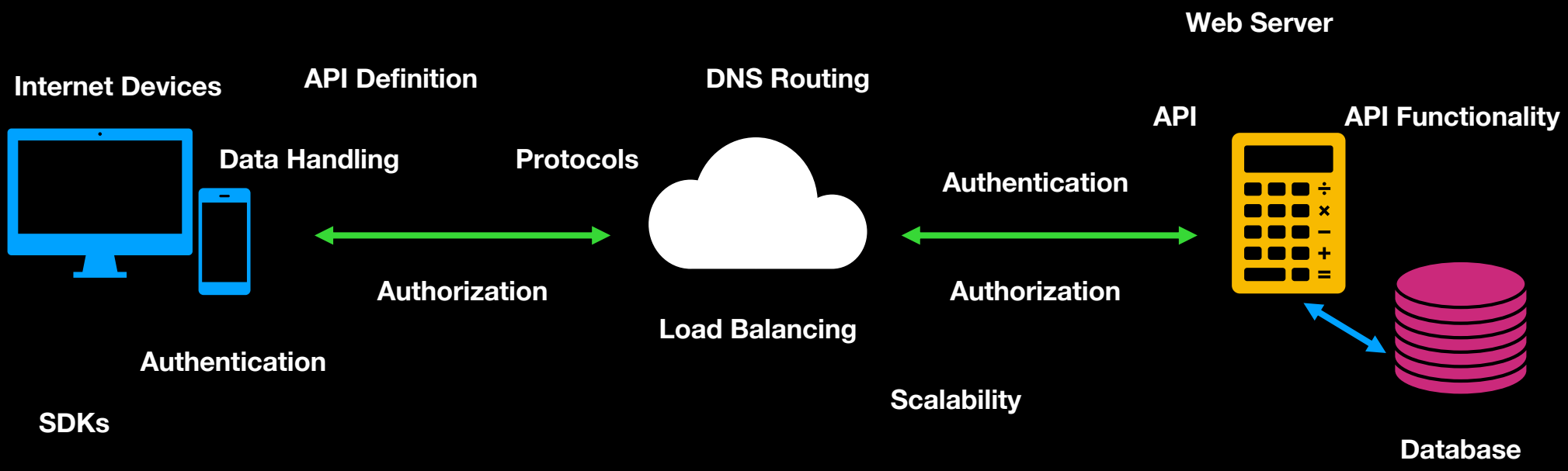


- **Create Github Account**
  https://github.com/signup

- **Optional: Install Nodejs/npm**
  https://nodejs.org/en/download/

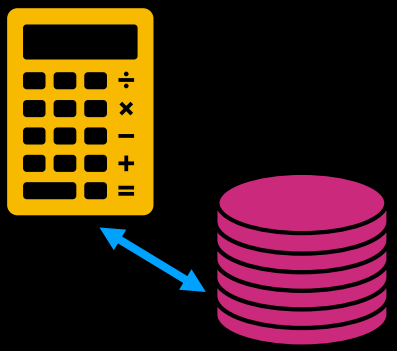
API Design: Introduction

API Design: Introduction



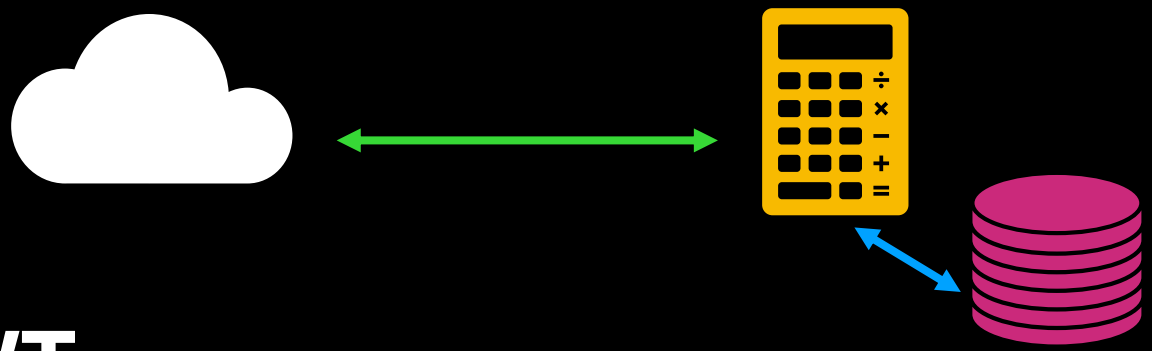
API Design: Introduction

- **API Backend**
  - **Servers (e.g., Apache, Tomcat)**
    - **Web Server Apps**
    - **Processes API Requests**
    - **Functionality**
      - **Java, C#, Python, PHP, etc.**
  - **Data**
    - **Database (e.g., Oracle, MySQL)**



API Design: Introduction

- **API Routing & Security**
  - **DNS**
  - **Load Balancing**
    - **Scalability**
  - **Authentication**
    - **OAuth, Token/JWT**
  - **Authorization**
    - **Permissions**



API Design: Introduction

- **API Client**

Device on Internet

Understands API

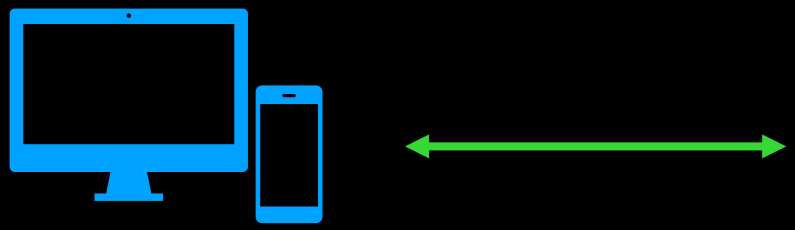
Endpoints (Nouns, Verbs)

Data formats (JSON)

Login/Authentication

e.g., Token Header

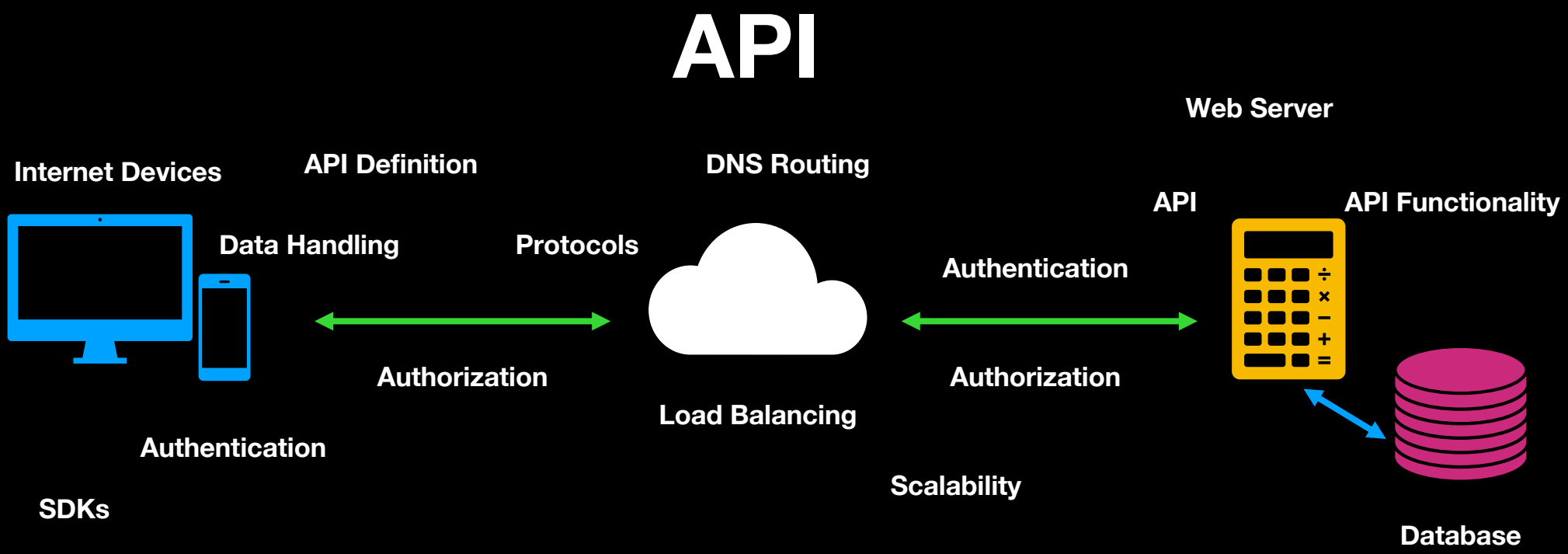
API Design: Introduction



- **Client Communication**
  - **HTTP**
  - **JSON, XML, etc.**
  - **API Endpoints**
  - **Provided SDKs**
    - **Generated code**

API Design: Introduction

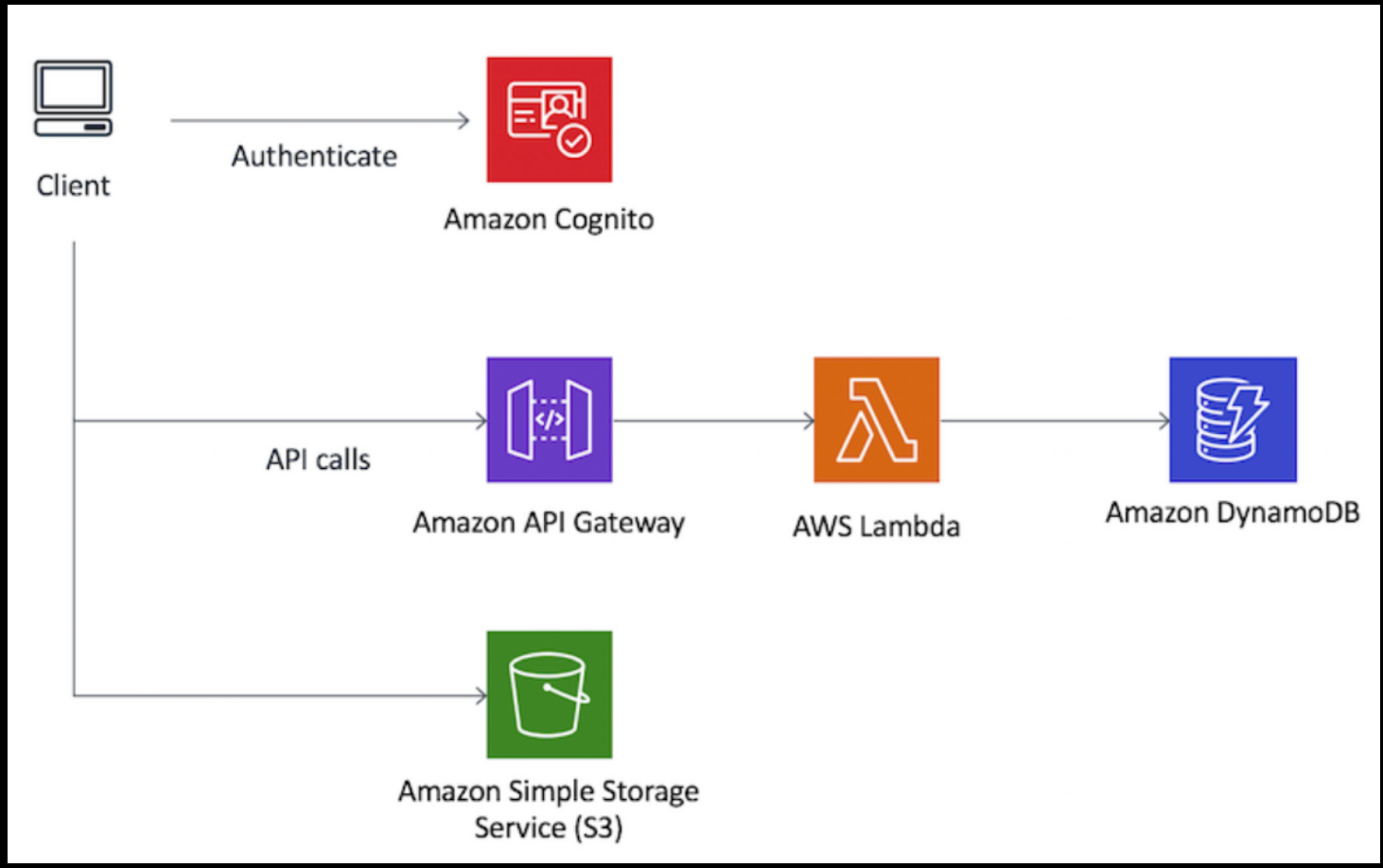
API Design: Introduction



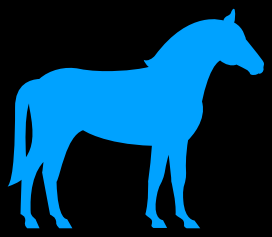
OpenAPI Spec

API Design: Introduction

- **AWS**



API Design: Introduction



- **Past:**

  API Wild West

  URL/Query String, Form Data, etc.

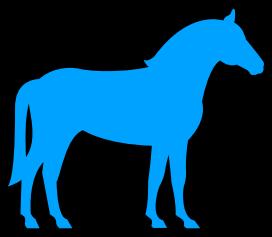
  SOAP/WSDL

  XML, text, etc.

Everything was proprietary, complex

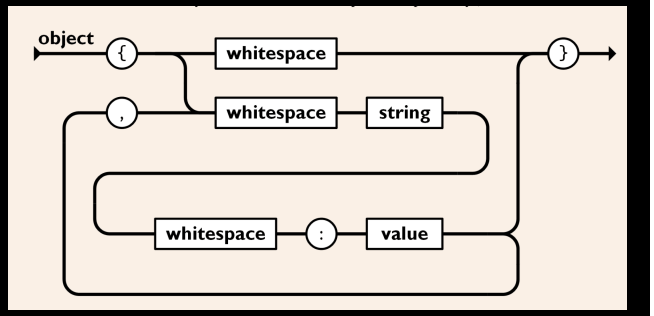
Needed a Standard

API Design: Introduction

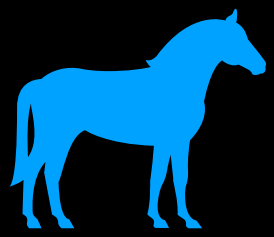


- **2000:**
  Roy Fielding's doctoral dissertation on REST (REpresentational State Transfer)

2001:
JSON by Douglas Crockford
JavaScript Object Notation
json.org



API Design: Introduction



Early 2000s - Money to be made

- **Access data**
- **Useful functionality, storage**

eBay, Flickr, Amazon, Twitter, etc.

- **The standard is key**

API Design: Introduction

- **REST (or other) API is expected now**

Libraries/frameworks/SDKs abound

Variations cause problems

Downside: better solutions will suffer from inertia

API Design: Introduction

gRPC - grpc.io

gRPC is a modern open source high performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

- g = ?
- Remote Procedure Call
- "a client application can directly call a method on a server application on a different machine as if it were a local object"

GraphQL

- flexible
- data efficient
- Facebook 2012, open-source 2015

```
{
  user(id: 4802170) {
    id
    name
    isViewerFriend
    profilePicture(size: 50)  {
      uri
      width
      height
    }
    friendConnection(first: 5) {
      totalCount
      friends {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "user": {
      "id": "4802170",
      "name": "Lee Byr
      "isViewerFriend"
      "profilePicture"
        "uri": "cdn://
        "width": 50,
        "height": 50
      },
      "friendConnectic
        "totalCount":
        "friends": [
          {
            "id": "305
            "name": "$
          },
          {
            "id": "310
            "name": "N
```

Image from: https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/

API Design: Introduction

- **Enter: OpenAPI**
  - **Standard**
  - **Versioned (e.g., 3.0.1)**

API OpenAPI spec
- **Provides details of implementation Contract/Standard**
- **Versioned (e.g., 1.1)**
- **Starts with Info section**

- **Editor: Info**

editor.swagger.io

Start with version

Errors

Insert > Info

API Design: Introduction

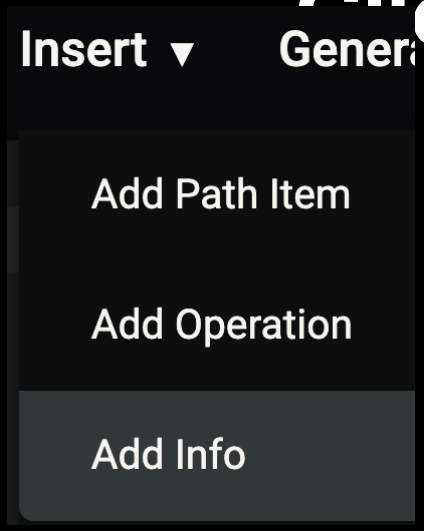
- **Editor**

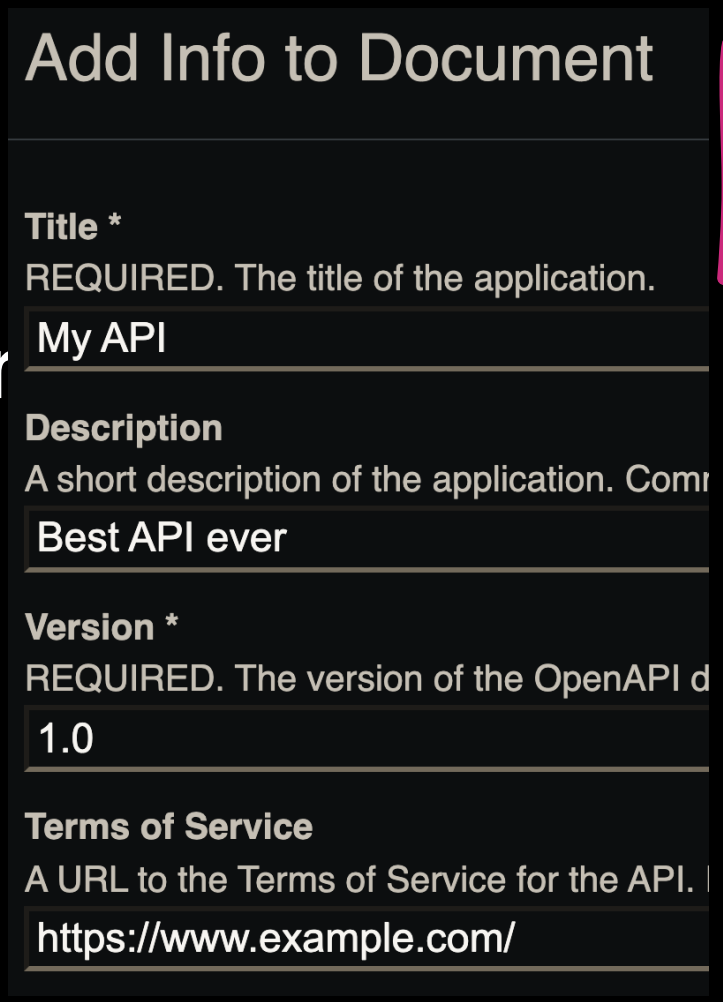
Insert > Info

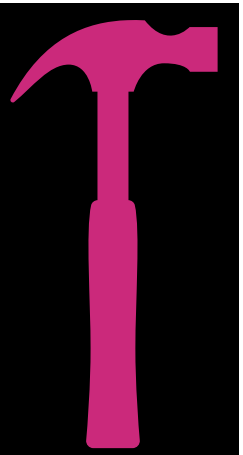
Enter text in fields

License/Contact optior

Click 'Add Info'







- **Text inserted**

```yaml
openapi: 3.0.1
info:
  title: My API
  version: '1.0'
  description: Best API ever
  termsOfService: https://www.example.com/
  contact:
    email: bear@brainwashinc.com
    name: Bear Cahill
    url: http://www.example.com/contact
  license:
    name: My License
    url: http://www.example.com/license
```

End of lab

API Design: JSON

- **RESTful JSON**

JSON

JavaScript Object Notation

json.org

Arrays, Dictionaries of values

Represented in text

Parseable across platforms, languages, etc.

```
value
    object
    array
    string
    number
    "true"
    "false"
    "null"

object
    '{' ws '}'
    '{' members '}'

members
    member
    member ',' members

member
    ws string ws ':' element

array
    '[' ws ']'
    '[' elements ']'
```

API Design: JSON

- **JSON**

Object Example:

User

userId: String
userName: String
isActive: Boolean
rank: Integer
address: Dictionary/Hash

API Design: JSON

- **JSON Object Example:**
  - **User**
    - **userId: String**
    - **userName: String**
    - **isActive: Boolean**
    - **rank: Integer**
    - **address: Dictionary/Hash**

```
{
    "userId": "ABC12341",
    "userName": "brainofbear",
    "isActive": true,
    "rank": 5,
    "address": {
        "street": "500 Main St.",
        "city": "Denton",
        "postal": 76201
    }
}
```

API Design: JSON

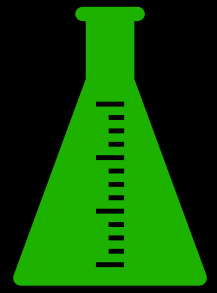
- JSON
  Object name (User) not included
  Arrays [ ]
  Dictionary { }
  Keys are Strings
  Values vary
    Strings, Ints, Bool, Dictionaries, Array

```
value
    object
    array
    string
    number
    "true"
    "false"
    "null"

object
    '{' ws '}'
    '{' members '}'

members
    member
    member ',' members

member
    ws string ws ':' element

array
    '[' ws ']'
    '[' elements ']'
```

API Design: JSON

- **JSON**
  RESTful JSON - very common combo
  SDKs often provide parsing
  Sometimes built into languages
    JS, Swift (Codeable)

API Design: JSON



- **Lab**

  https://jsonlint.com/

  Create an array of Elements

  Each element has 3 members:

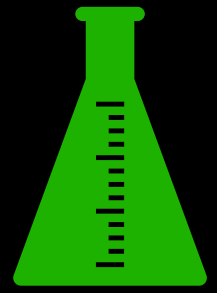
  name of type string

  description of type string

  age of type integer

  See next slide for hints

API Design: JSON



- **Lab**

json.org has details

Arrays are within [and]

Elements are within { and }

Element members are key-value pairs

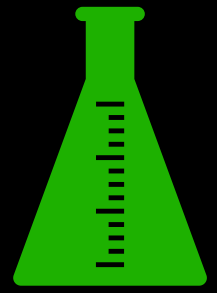
"name" : "test"

Members and Array elements are comma separated

See next slide for example

API Design: JSON

- **Lab**



```
[
  {
    "name":"thing 1",
    "description":"just a thing",
    "age":4
  },
  {
    "name":"thing 2",
    "description":"just another thing",
    "age":10
  },
  {
    "name":"thing 3",
    "description":"just a third thing",
    "age":25
  }
]
```

End of lab

API Design: Design

- **Key Concepts**

HTTP for communication

State is…

Value(s) of data at a given time

Cookies, Parameters, etc. represent state

REST Statelessness

Nothing stored by server (or possibly even sent by client) about state

API Design: Design

Key Aspects

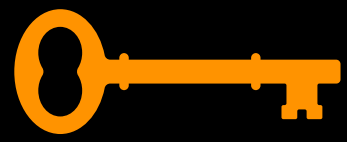
- **Nouns - resource based**
  - **e.g., User, Comment, Post**
- **Verbs to URIs**
  - **e.g., GET, POST**
- **Stateless - each request has it all**
  - **Scaleable - doesn’t need same server for next request**

API Design: Design

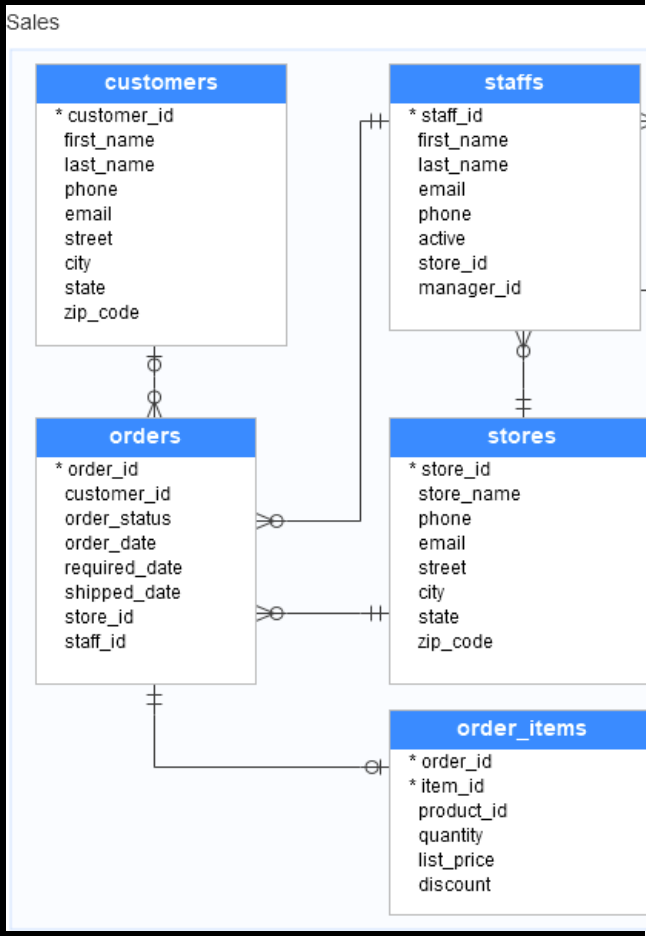
Key Aspects

- **Nouns - resource based**
  - **e.g., Users**
    - **<domain>/users**
    - **http://www.example.com/api/users**
      - **Array of User objects (JSON): [ ]**
    - **<domain>/users/<id>**
    - **http://www.example.com/api/users/234**
      - **1 User object: { }**

API Design: Design



- **Key Aspects**
  - **Example: DB Tables**
    - **…/customers**
      - **…/customers/{customer_id}**
    - **…/staffs**
    - **…/orders**
    - **…/stores**
    - **…/orderItems**
  - **Could be other resources**



API Design: Design

- **Key Aspects**
  - **Nouns - resource based**
    - **Relationships in URL**
      - **Comments relates to Post 1**
      - **/posts/1/comments - returns 2 comments**
      - **/comments/1/posts - returns 1 post**

Data Source ->

```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ],
```

API Design: Design

Verbs - CRUD

- POST = Create
- GET = Read
- PATCH = Update
- DELETE = Delete

Data Source ->


```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ],
```

API Design: Design

Verbs - CRUD

- **GET - read**
  - **.../posts - all**
  - **.../posts/1 - post with id = 1**
  - **.../comments - all**
  - **.../comments/1 - one**
  - **.../posts/1/comments - comments related to post via postId**

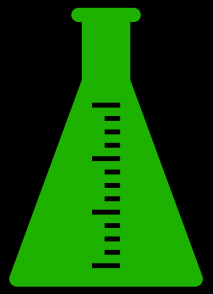
Data Source ->


```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ],
```

Lab: GET

API Design: Design

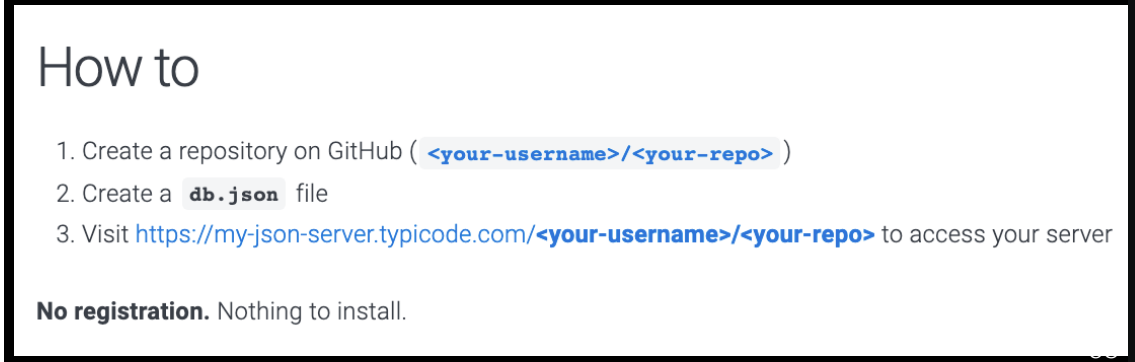
https://my-json-server.typicode.com/typicode/demo

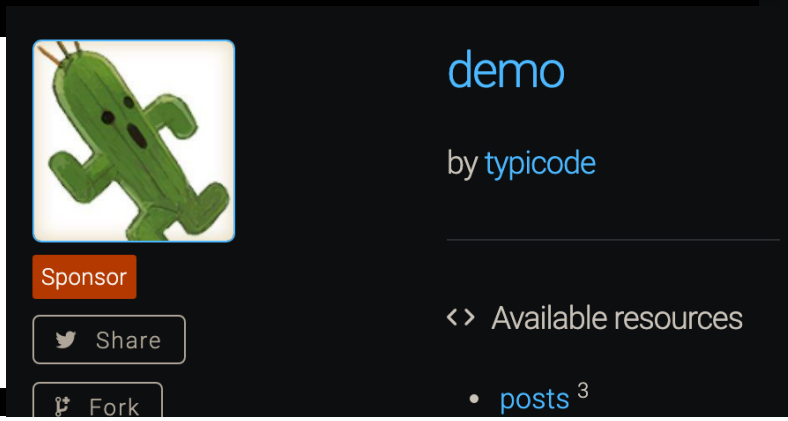


View db.json file

Via browser:

GET posts, single post, comments, single comment, comments for 1 post





API Design: Design

- Verbs - Create, Update
  - POST = Create
  - PATCH = Update
  - PUT
    - If exists, replace
    - Otherwise, create
    - Idempotent - multiple requests, same result

```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ],
```

API Design: Design

- **Verbs - CRUD**
  - **POST = Create**
  - **GET = Read**
  - **PATCH = Update**
  - **DELETE = Delete**
- **POST/PUT/PATCH**
  - **Overlap**
  - **Require HTTP body**


```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ],
```

API Design: Design

Query String

- **Additional parameters:**
  - **Filtering**
  - **Searching**
  - **Paging**
  - **Embedding**
  - **etc.**

…/api/posts/1?_embed=comments

```
{
  "id": 1,
  "title": "Post 1",
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ]
}
```

API Design: Design

- All defined in…

OpenAPI Spec

Contract Description

Information

Servers

Paths & Methods

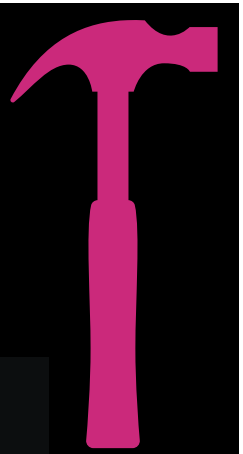
Model

Security

All Interested Parties

Field Name	Type
openapi	string
info	Info Object
servers	[Server Object]
paths	Paths Object
components	Components Object
security	[Security Requirement Object]
tags	[Tag Object]
externalDocs	External Documentation Object

```yaml
swagger: "2.0"
info:
  description: "This is a sample server Petstore serve
    can find out more about     Swagger at [http://swa
    (http://swagger.io) or on [irc.freenode.net, #swag
    ://swagger.io/irc/).      For this sample, you can
    api key `special-key` to test the authorization
    ."
  version: "1.0.0"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  contact:
    email: "apiteam@swagger.io"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.h
host: "petstore.swagger.io"
basePath: "/v2"
tags:
- name: "pet"
  description: "Everything about your Pets"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
- name: "store"
  description: "Access to Petstore orders"
- name: "user"
  description: "Operations about user"
  externalDocs:
    description: "Find out more about our store"
    url: "http://swagger.io"
schemes:
- "https"
- "http"
paths:
  /pet:
    post:
      tags:
      - "pet"
      summary: "Add a new pet to the store"
      description: ""
      operationId: "addPet"
```

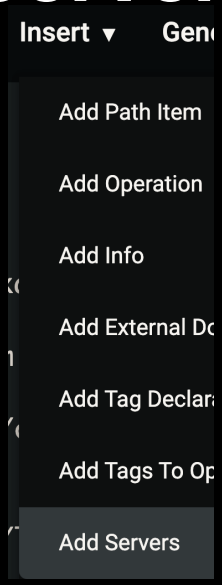


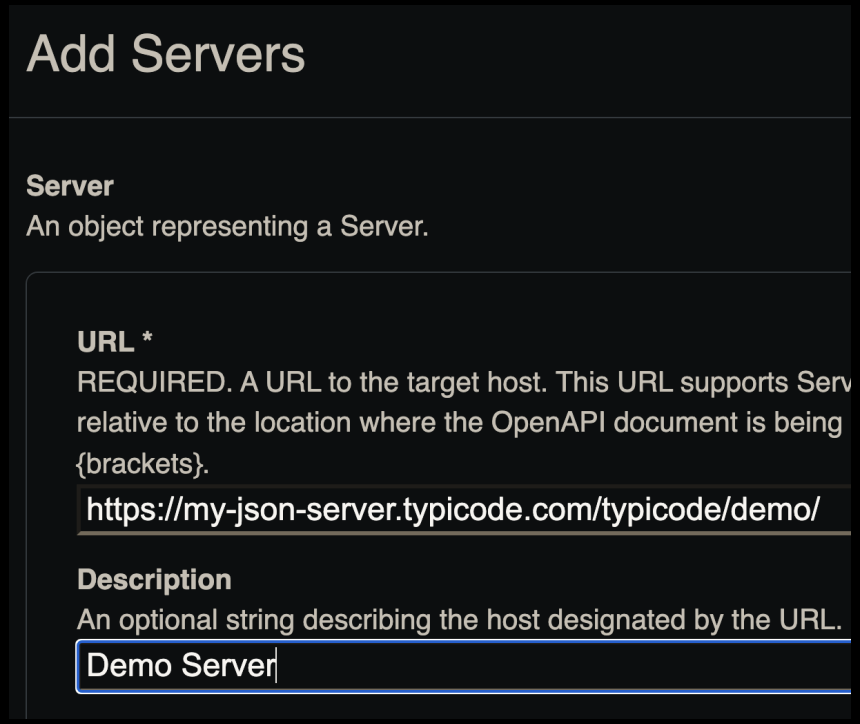
- **Now that we have one…**

Add Server

editor.swagger.io

Insert > Add Servers



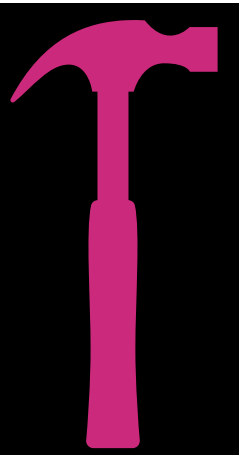


Add:

https://my-json-server.typicode.com/typicode/demo/

API Design: Design Concepts

- **Server**

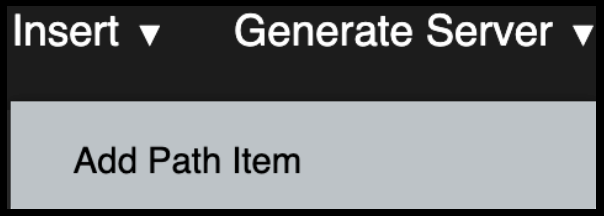


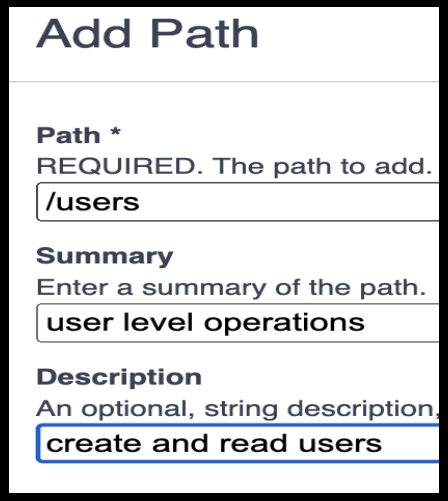
```yaml
openapi: 3.0.1
info:
  title: My API
  version: '1.0'
  description: Best API ever
  termsOfService: https://www.example.com/
  contact:
    email: bear@brainwashinc.com
    name: Bear Cahill
    url: http://www.example.com/contact
  license:
    name: My License
    url: http://www.example.com/license
servers:
  - url: https://my-json-server.typicode.com/typicode/demo/
    variables: {}
    description: Demo Server
```

End of Lab

API Design: Design Concepts

- **Lab: OpenAPI**
- **Add Path Insert > Add Path Item**
- **Fill in form**
- **Click Add Path**



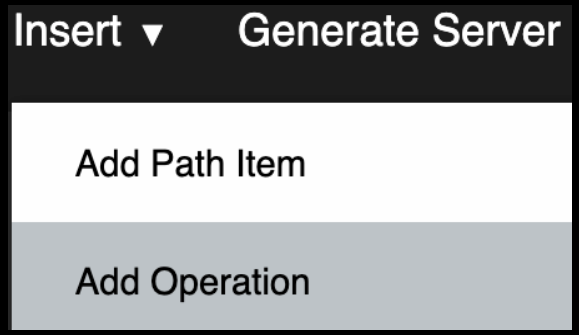


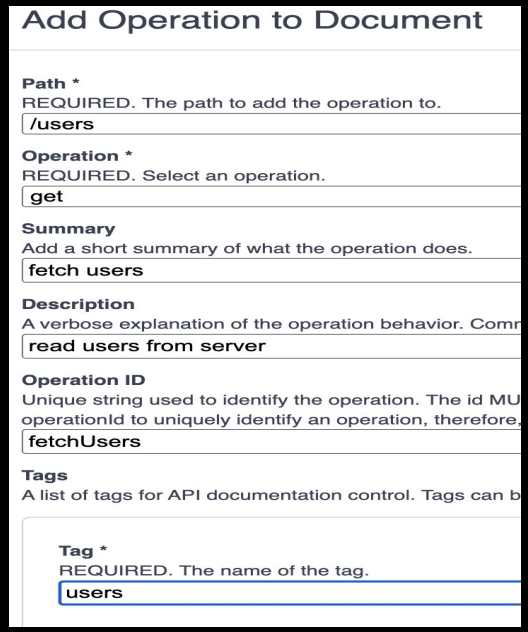
Creates:

```
paths:
  /users:
    summary: user level operations
    description: create and read users
```

API Design: Design Concepts

- **Lab: OpenAPI Add Path Method: GET**
- **Insert > Add Operation**
- **Fill in form**
- **Click Add Operation**



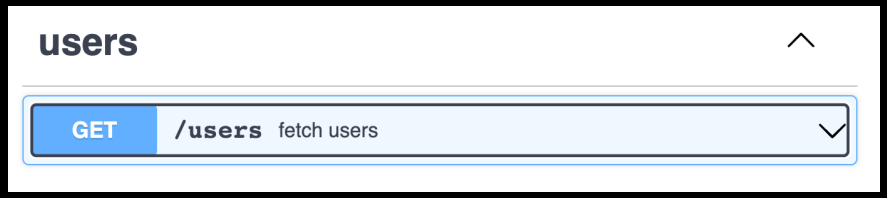


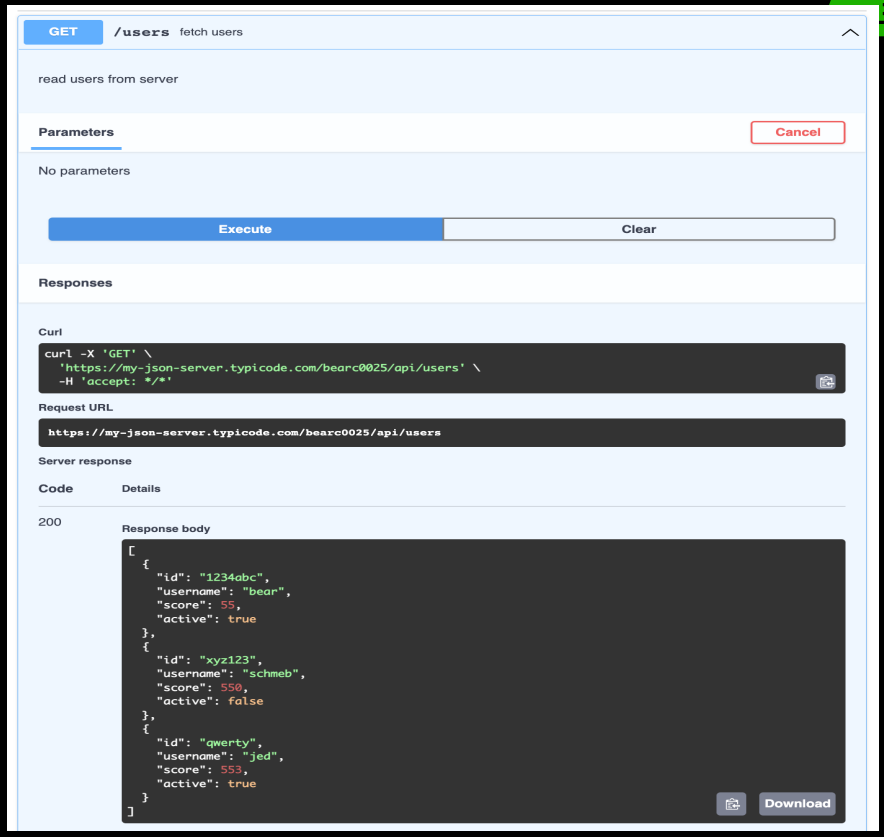
Creates:

```yaml
paths:
  /users:
    summary: user level operations
    description: create and read users
    get:
      summary: fetch users
      description: read users from server
      operationId: fetchUsers
      responses:
        default:
          description: Default error sample response
      tags:
        - users
```

API Design: Design Concepts

- **Lab: OpenAPI Add Path Method: GET**
- **Notice updated documentation**
- **Execute GET**





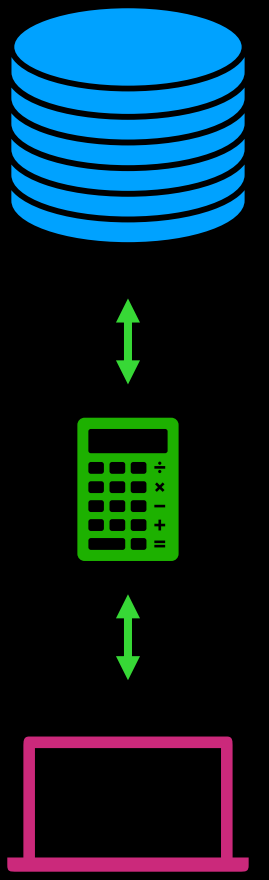
- **End of lab**

For more… PluralSight

APIs- Application Programming Interface: Executive Briefing

Dan Appleman

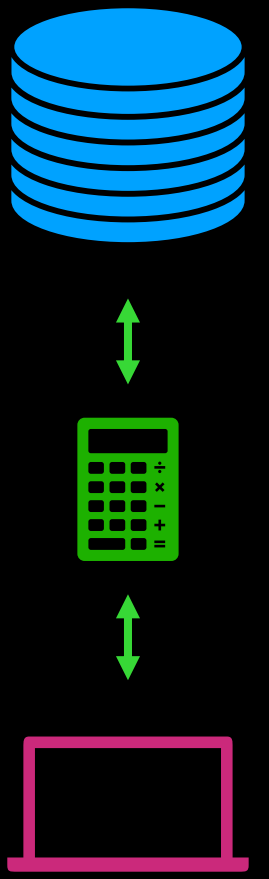
API Design: Design Concepts



API Design: Design Concepts

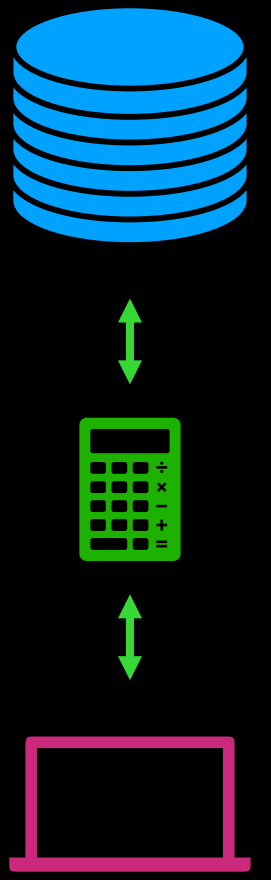
Separation of Concerns

- **Gathering Data - e.g., from database**
- **Formatting Data - e.g., JSON**
- **Delivering Data - e.g., HTTP, streaming**
- **Security**
- **Scaleability**
- **Reliability**
- **Etc.**



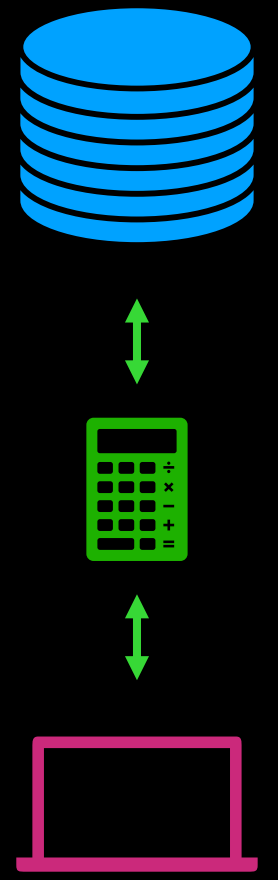
API Design: Design Concepts

- **Separation of Concerns**
  - **Gathering Data**
    - **When requested by client**
    - **Very important on the server**
    - **Good database design**
    - **Cache if possible**
    - **Efficient queries**

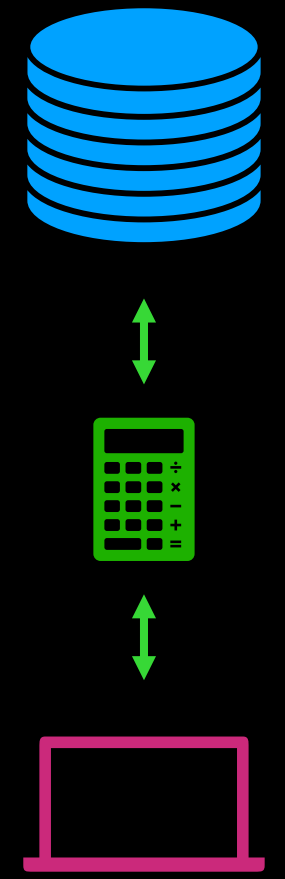


Separation of Concerns

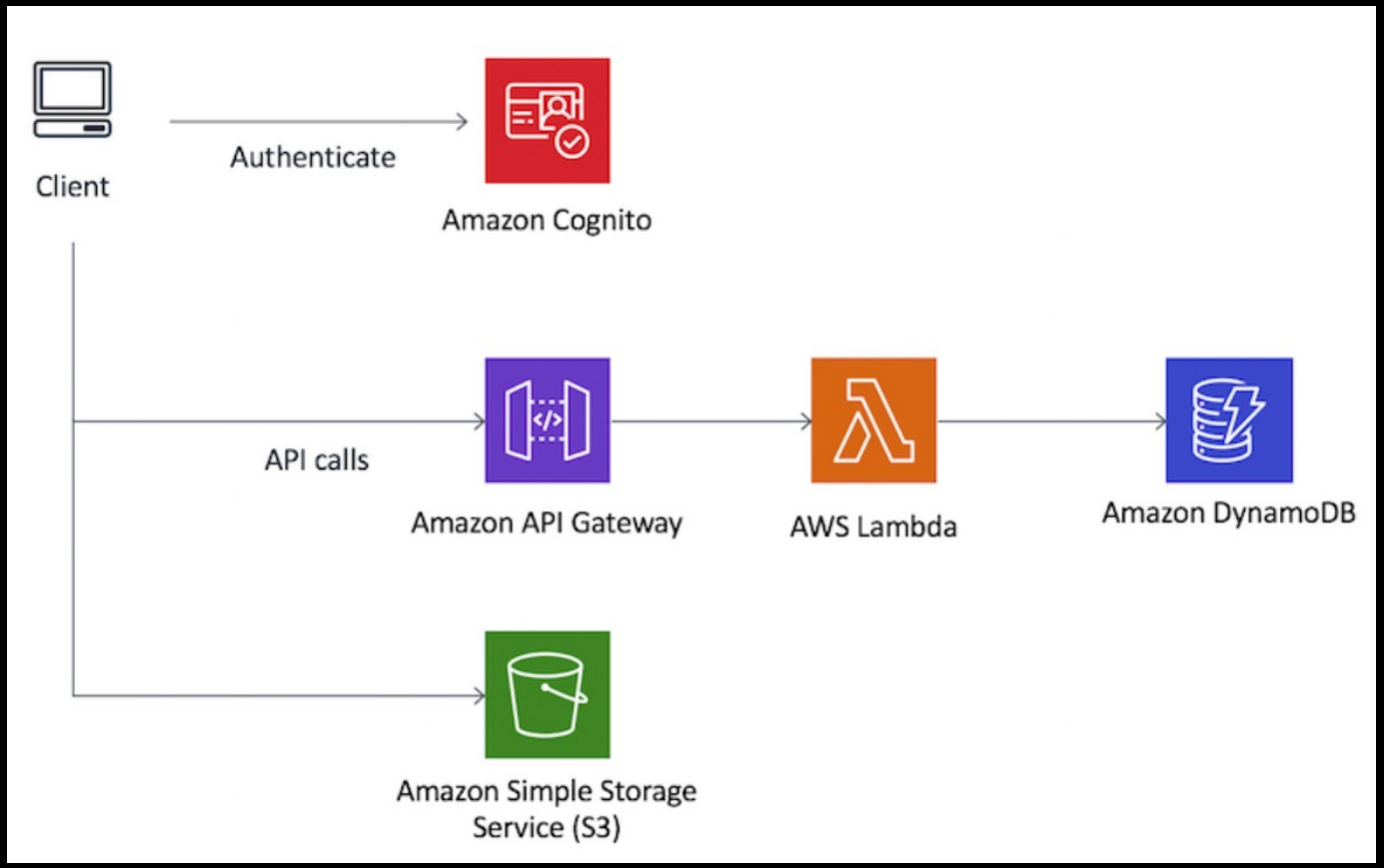
- **Formatting Data**
  - **When sending/returning to client**
  - **Important to client**
  - **Directed by server (one-for-all)**

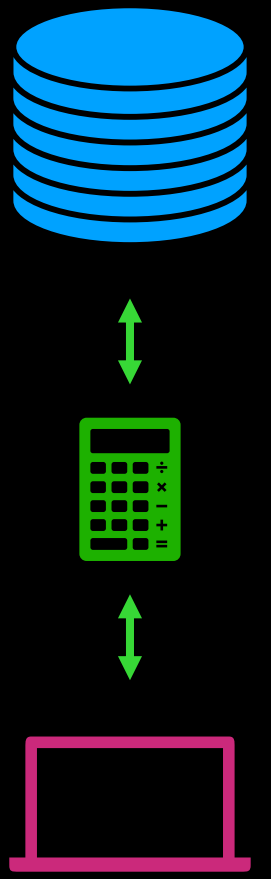


- Separation of Concerns
  - Delivering Data
    - How it is sent: protocol, etc.
    - Important to client
    - Directed by server
      - Usually the same for all clients



AWS - Separation of Concerns

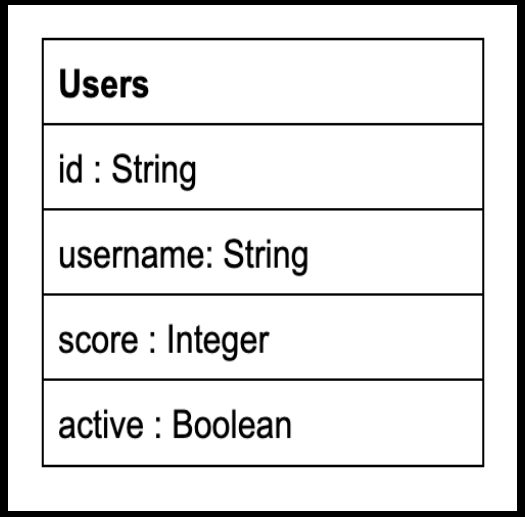




- Operations: Responses
  - Default Response
  - Specified by Status Code (e.g., 200, 404, etc.)
  - Includes:
    - Description
    - Content
      - Content Type
        - Schema

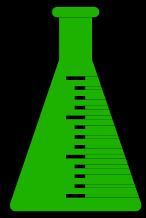
```yaml
get:
  summary: fetch users
  description: read users from server
  operationId: fetchUsers
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```

- For all 200-299 response
- `application/json` content
- Schema is an object
- With properties

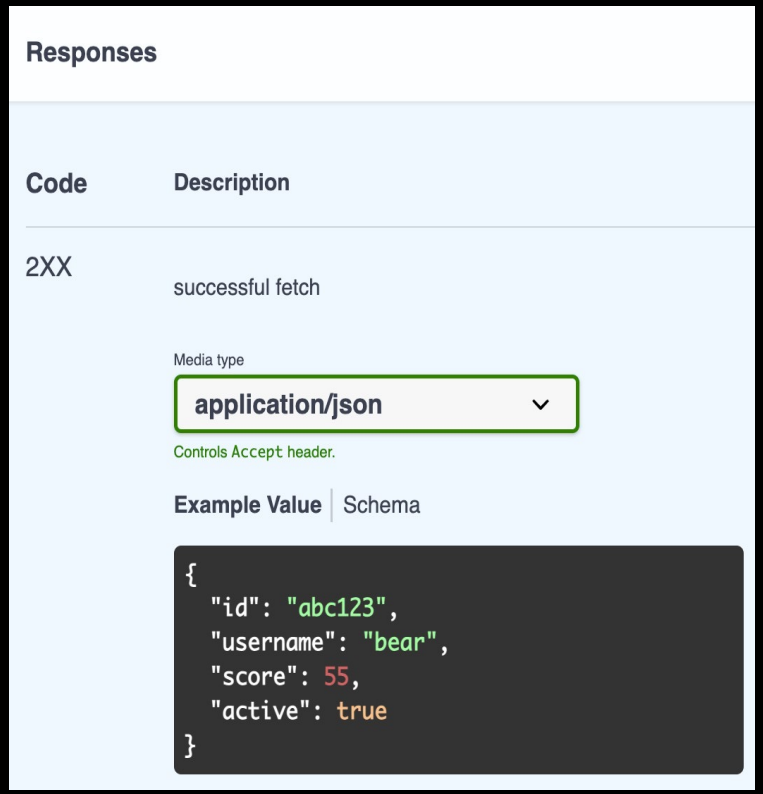



```yaml
get:
  summary: fetch users
  description: read users from server
  operationId: fetchUsers
  responses:
    '2XX':
      description: successful fetch
      content:
        application/json:
          schema:
            type: object
            properties:
              id:
                type: string
                example: "abc123"
              username:
                type: string
                example: "bear"
              score:
                type: integer
                example: 55
              active:
                type: boolean
                example: true
```

- Lab: OpenAPI Add GET Response
- NOTE: We'll make this an array later.



```yaml
get:
  summary: fetch users
  description: read users from server
  operationId: fetchUsers
  responses:
    '2XX':
      description: successful fetch
      content:
        application/json:
          schema:
            type: object
            properties:
              id:
                type: string
                example: "abc123"
              username:
                type: string
                example: "bear"
              score:
                type: integer
                example: 55
              active:
                type: boolean
                example: true
```



- End of lab

Errors

- **HTTP**
- **Response Codes**
  - **1xx: Informational**
  - **2xx: Success**
  - **3xx: Redirection**
  - **4xx: Client Error**
  - **5xx: Server Error**

https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Errors - Best Practices

- **Use the correct codes**
- **Use the right code granularity (not every single status code)**
- **Consider the response message depth - e.g., do you specify the first thing that's wrong with the request or everything?**
- **Security - don't give away secrets**

Errors - Best Practices

- "Helpful" information is best
  - Readable message
  - Meaningful codes (http and internal)
  - Hints/Details when possible and appropriate

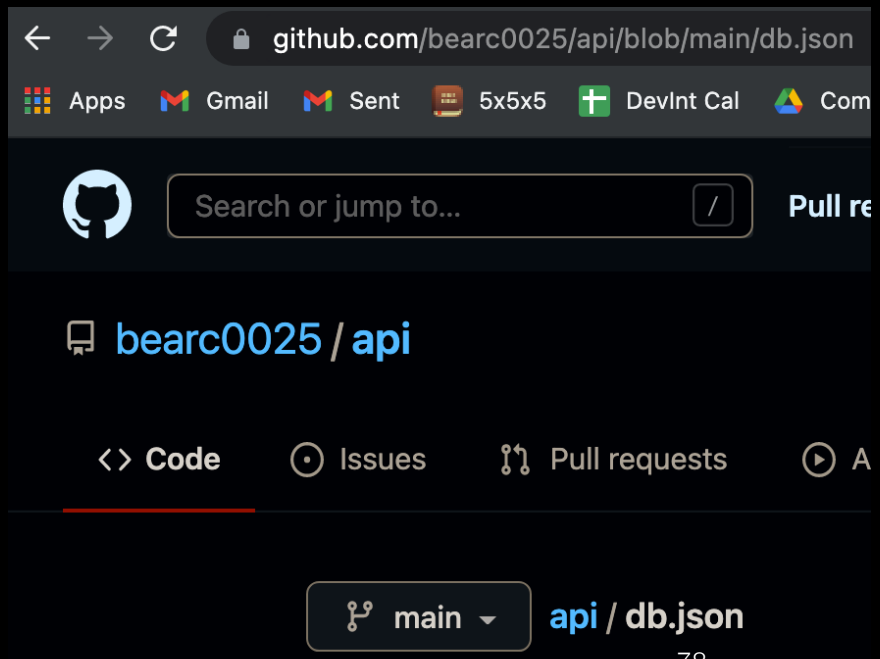
API Design: Design Concepts

- **Environment**
  - **Development**
  - **Staging**
  - **Production**
  - **Other**
  - **Internal/External**
- **OpenAPI supports multiple servers**

Demo:

GitHub Repo with db.json file

github.com/bearc0025/api/blob/main/db.json




```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment"
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment"
      "postId": 1
    }
  ],
  "profile": {
    "name": "typicode"
  }
}
```

API Design: Design

Demo:

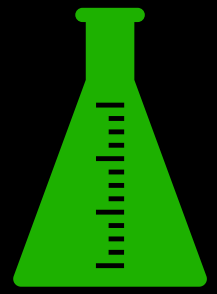
- https://my-json-server.typicode.com/bearc0025/api
- https://my-json-server.typicode.com/bearc0025/api/posts
- https://my-json-server.typicode.com/bearc0025/api/posts/1?_embed=comments


```
{
  "posts": [
    {
      "id": 1,
      "title": "Post 1"
    },
    {
      "id": 2,
      "title": "Post 2"
    },
    {
      "id": 3,
      "title": "Post 3"
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
```



```
{
  "id": 1,
  "title": "Post 1",
  "comments": [
    {
      "id": 1,
      "body": "some comment",
      "postId": 1
    },
    {
      "id": 2,
      "body": "some comment",
      "postId": 1
    }
  ]
}
```

API Design: Design



Lab

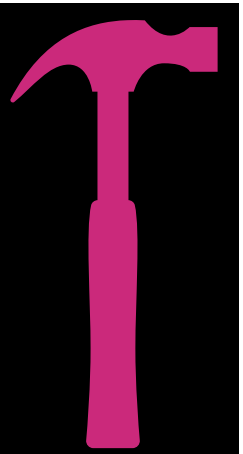
If you have a git repo…

https://my-json-server.typicode.com/

Demo data @ https://my-json-server.typicode.com/typicode/demo/db

How to

1. Create a repository on GitHub (`<your-username>/<your-repo>`)
2. Create a `db.json` file
3. Visit https://my-json-server.typicode.com/**<your-username>/<your-repo>** to access your server

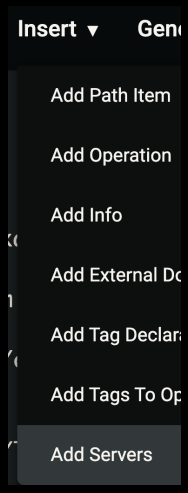


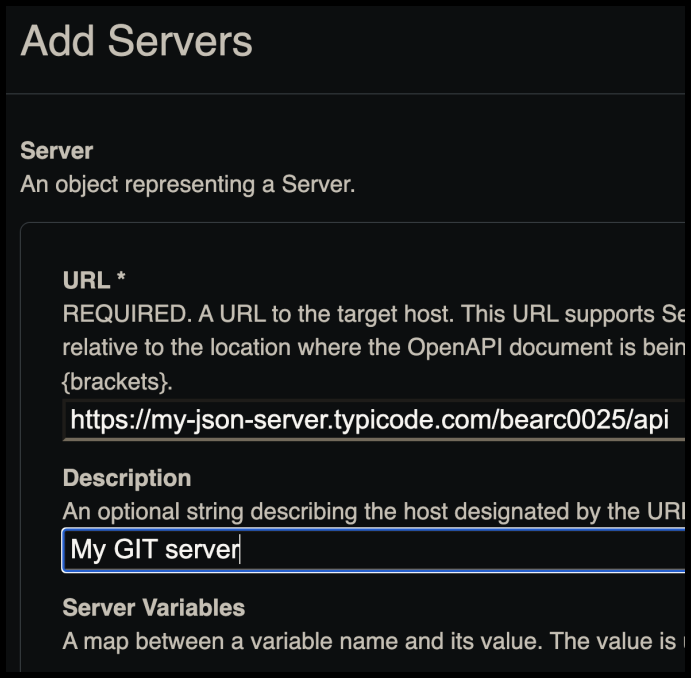
- **Now that we have two…**

Add Another Server

editor.swagger.io

Insert > Add Servers

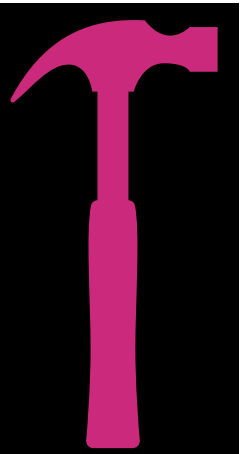




Add your server or mine:

https://my-json-server.typicode.com/bearc0025/api

- **Server**



```yaml
servers:
  - url: https://my-json-server.typicode.com/typicode/demo/
    variables: {}
    description: Demo Server
  - url: https://my-json-server.typicode.com/bearc0025/api
    variables: {}
    description: My GIT server
```

End of Lab

API Design: Richardson Maturity Model

API Design: Richardson Maturity Model

- **Breakdown https://martinfowler.com/articles/richardsonMaturityModel.html**

Four Levels:

Level 0: Uses HTTP

Level 1: Resources (nouns)

Level 2: HTTP Verbs

Level 3: Hypermedia Controls (HATEOAS = Hypertext As The Engine Of Application State)

API Design: Richardson Maturity Model

- **Level 0: Uses HTTP**
  - **Mostly just HTTP plus anything…**
    - **XML, YAML, JSON, etc.**
    - **RPC**
  - **Might be one endpoint for all requests**
  - **Content specifies details for request and response**
  - **And/or GET query string**

API Design: Richardson Maturity Model

- **Level 1: Resources (nouns)**
  Various endpoints
    e.g., /posts, /posts/1
  Body/GET still specifies details
    Request, Response
  Various formats: XML, etc.

API Design: Richardson Maturity Model

- **Level 2: HTTP Verbs**

GET

Same request, same results

Cacheable

POST (create)

201 Created

409 Conflict

PUT, PATCH, DELETE, etc.

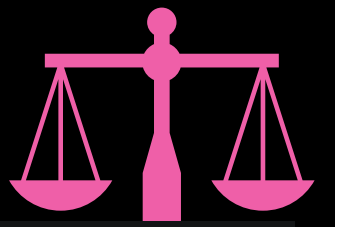
API Design: Richardson Maturity Model

- **Level 3: Hypermedia Controls (HATEOAS = Hypertext As The Engine Of Application State)**

What to do next and how

Can be dynamic

Client doesn’t need knowledge



```
{
  "departmentId": 10,
  "departmentName": "Administration",
  "locationId": 1700,
  "managerId": 200,
  "links": [
    {
      "href": "10/employees",
      "rel": "employees",
      "type" : "GET"
    }
  ]
}
```

API Design: Richardson Maturity Model

- **HATEOAS**

No official OpenAPI format

Common practice includes:

href: URI

rel: relation type

type: expected media resource type (e.g., JSON or https://restfulapi.net/hateoas/)

See RFC5988 and Hypertext Application Language (HAL)

```
{
  "departmentId": 10,
  "departmentName": "Administration",
  "locationId": 1700,
  "managerId": 200,
  "links": [
    {
      "href": "10/employees",
      "rel": "employees",
      "type" : "GET"
    }
  ]
}
```

API Design: Richardson Maturity Model

- **OpenAPI 3.0**
  Added links section in responses
  operationId or operationRef
  parameters or requestBody
  Optional: server and description

```yaml
responses:
  '201':
    description: Created
    content:
      application/json:
        schema:
          type: object
          properties:
            id:
              type: integer
              format: int64
              description: ID of the created user.
    # --------------------------------------------------
    # Links
    # --------------------------------------------------
    links:
      GetUserByUserId:   # <---- arbitrary name for the link
        operationId: getUser
        # or
        # operationRef: '#/paths/~1users~1{userId}/get'
        parameters:
          userId: '$response.body#/id'

        description: >
          The `id` value returned in the response can be used as
          the `userId` parameter in `GET /users/{userId}`.
    # --------------------------------------------------
```

API Design: Richardson Maturity Model

- operationId or operationRef specifies the related method of the link
  - operationRef by path (internal / escaped w/ ~1)
  - parameters based on referenced operation (name) and properties

```yaml
responses:
  '201':
    description: Created
    content:
      application/json:
        schema:
          type: object
          properties:
            id:
              type: integer
              format: int64
              description: ID of the created user.
    links:
      GetUserByUserId:
        description: >
          The `id` value returned in the response can be used as
          the `userId` parameter in `GET /users/{userId}`.
        operationId: getUser
        # or
        # operationRef: '#/paths/~1user~1{userId}/get'
        parameters:
          userId: $response.body#/id

/user/{userId}:
  get:
    summary: Get a user by ID
    operationId: getUser
    parameters:
      - in: path
        name: userId
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: A User object
```

API Design: Richardson Maturity Model

- **Links can be defined in the components section**

Links can be reused

e.g., /users and /users/{userId}

```yaml
          links:
            GetUserByUserId:
              $ref: '#/components/links/GetUserByUserId'    # <-------

    get:
      summary: Get a user by ID
      operationId: getUser
      ...

components:
  links:
    GetUserByUserId:   # <----- The $ref's above point here
      description: >
        The `id` value returned in the response can be used as
        the `userId` parameter in `GET /users/{userId}`.
      operationId: getUser
      parameters:
        userId: '$response.body#/id'
```

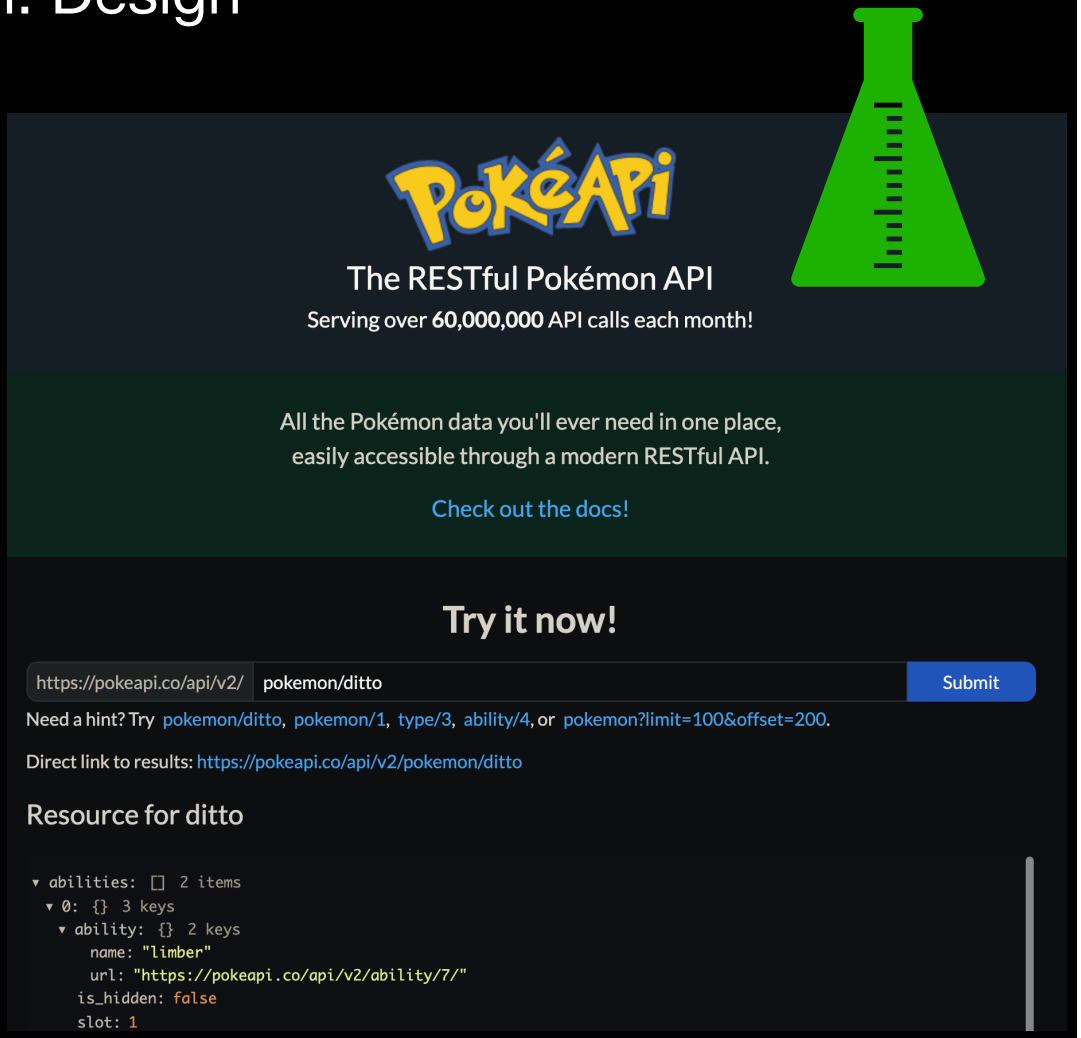
API Design: Design

Lab

https://pokeapi.co/

Fetch data and drill down into more

Try with limit and offset



API Design: Design

- **Lab**

https://swapi.dev/

Fetch data and drill down into more

Try with page query string parameter

https://swapi.dev/api/	people?page=5

API Design: Dev Standards

API Design: Dev Standards

- **Contract communicates details**
- **Maintaining the contract helps keep all parties consistent/correct**
- **Changes should be driven via the contract**
  - **Centralized update for all**

API Design: Dev Standards

- **API Spec**
  - **Swagger/OpenAPI**
  - **Endpoints (paths)**
  - **Methods (verbs)**
  - **Requests bodies**
  - **Response bodies**
  - **Model**
  - **Parameters**

API Design: Dev Standards

- **Benefits**
  - **Source of Truth**
  - **Contract with agreement**
  - **Communication**
  - **Reliable, dependable, predictable**
- **Otherwise - opposite of above**

- **Remember: An API is meant to be used - don’t make it difficult**

API Design: Dev Standards

- **Richardson Maturity Model**
  - **Use HTTP, resources, verbs and (possibly) Hypermedia**
- **Security - OAuth, SSL**
- **Documentation**
  - **Readily available**
  - **Updated**
- **Versioning - in URL, header, both**

API Design: Dev Standards

- **Filtering/Sorting/Searching/Limiting/Pagination**
  - **Query string parameters**
  - **Route alias for complex/common**
    - **/posts/most_recent**
  - **Embed/expand**
- **Caching**
  - **Custom, ETag (hash version of resource) or Last-Modified**
  - **304 Not Modified response**

API Design: Dev Standards

- POST/PUT/PATCH - return resource
- JSON
  - Including PUT/POST/PATCH http body
  - Not key-value: name=bear&ver=1.1
  - Content-type: application/json
- Camel case vs Snake case
  - Camel for JavaScript and similar
  - camelCase vs snake_case

API Design: Dev Standards

- **HEAD**
  - **Identical to GET but no body for response**
  - **Meta info in headers per resource**
    - **Last mod, length, content type**
  - **Can be used for URL validity tests**
  - **Cacheable**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Type: text/html; charset=UTF-8
Date: Wed, 08 May 2013 10:12:29 GMT
ETag: "780602-4f6-4db31b2978ec0"
Last-Modified: Thu, 25 Apr 2013 16:13:23 GMT
Content-Length: 1270
```

- **OPTIONS**
  - **API options (Allow/Content-type)**
  - **Doesn't initiate data fetch/storage**
  - **May apply to server or specific resource**
  - **Non-Cacheable**
    - **It's not a true resource (GET)**
    - **https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html**

```
HTTP/1.1 200 OK
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: text/html; charset=UTF-8
Date: Wed, 08 May 2013 10:24:43 GMT
Content-Length: 0
```

9.2 OPTIONS

The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Responses to this method are not cacheable.

- **TRACE**
  - **Should reflect the message received back to client as the body**
  - **Allows client to see what/how is being received on the other end**

- **See https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html**

API Design: Dev Standards

- **Error handling**
  - **Thorough and consistent**
  - **HTTP status codes**
  - **Unique codes**
  - **Message for user**
  - **Message for log**

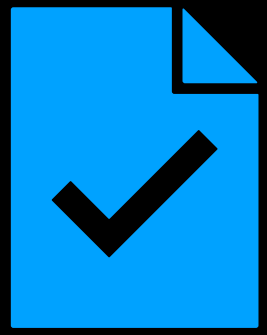
API Design: Contracts

Describes/Defines the API

OpenAPI 3.0 (Swagger tools)

JSON, YAML and others

Includes info, server, paths, etc.



```yaml
openapi: 3.0.1
info:
  title: Swagger Petstore
  description: 'This is a sample server Pe
    find out more about
    Swagger at [http://swagger.io](http://
      .freenode.net, #swagger](http://swag
    this sample, you can use the api key `
      the authorization filters.'
  termsOfService: http://swagger.io/terms/
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LI
  version: 1.0.5
```

```yaml
servers:
- url: https://petstore.swagger.io/v2
- url: http://petstore.swagger.io/v2
```

```yaml
paths:
  /pet:
    put:
      tags:
      - pet
      summary: Update an exis
      operationId: updatePet
```

API Design: Contracts

OpenAPI Documentation

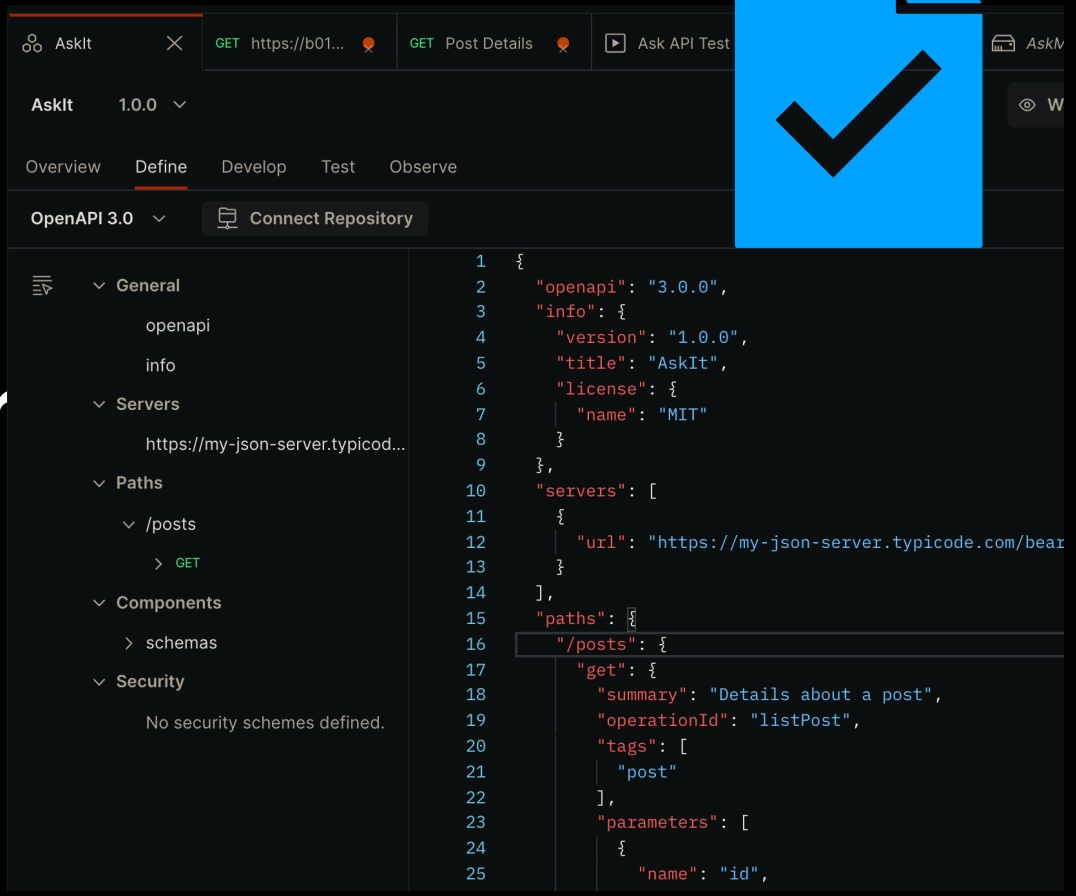
https://swagger.io/specification/

Various sections and types

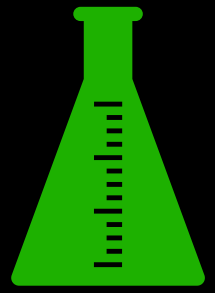
Field Name	Type	Description
openapi	`string`	**REQUIRED**. This string MUST be the semantic version number of the OpenAPI Specification version that the OpenAPI document uses. The `openapi` field SHOULD be used by tooling specifications and clients to interpret the OpenAPI document. This is *not* related to the API `info.version` string.
info	Info Object	**REQUIRED**. Provides metadata about the API. The metadata MAY be used by tooling as required.
servers	[Server Object]	An array of Server Objects, which provide connectivity information to a target server. If the `servers` property is not provided, or is an empty array, the default value would be a Server Object with a url value of `/`.
paths	Paths Object	**REQUIRED**. The available paths and operations for the API.
components	Components Object	An element to hold various schemas for the specification.
security	[Security Requirement Object]	A declaration of which security mechanisms can be used across the API. The list of values includes alternative security requirement objects that can be used. Only one of the security requirement objects need to be satisfied to authorize a request. Individual operations can override this definition. To make security optional, an empty security requirement (`{}`) can be included in the array.
tags	[Tag Object]	A list of tags used by the specification with additional metadata. The order of the tags can be used to reflect on their order by the parsing tools. Not all tags that are used by the Operation Object must be declared. The tags that are not declared MAY be organized randomly or based on the tools' logic. Each tag name in the list MUST be unique.
externalDocs	External Documentation Object	Additional external documentation.

API Design: Contracts

- Tools
  - SwaggerUI/Hub
  - Insomnia
  - OpenAPI Generator
  - Postman
  - PAW
  - Stoplight
  - Swashbuckle
- Can help generate or consume

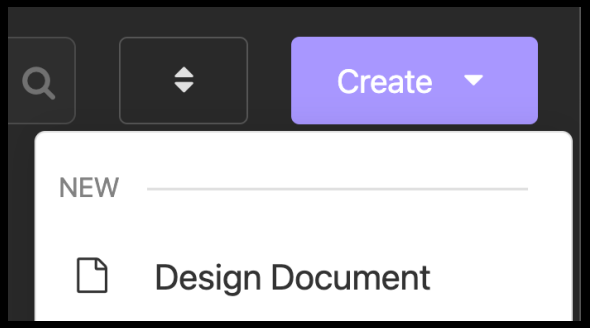


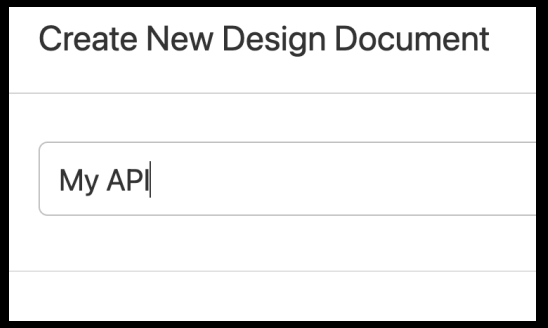
API Design: Code First



Create API in Insomnia

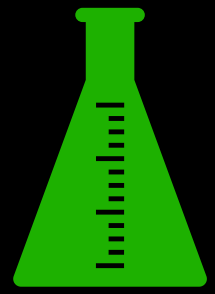
- Open Insomnia
- Click Create > Design Document
  - Give it a name and click Create

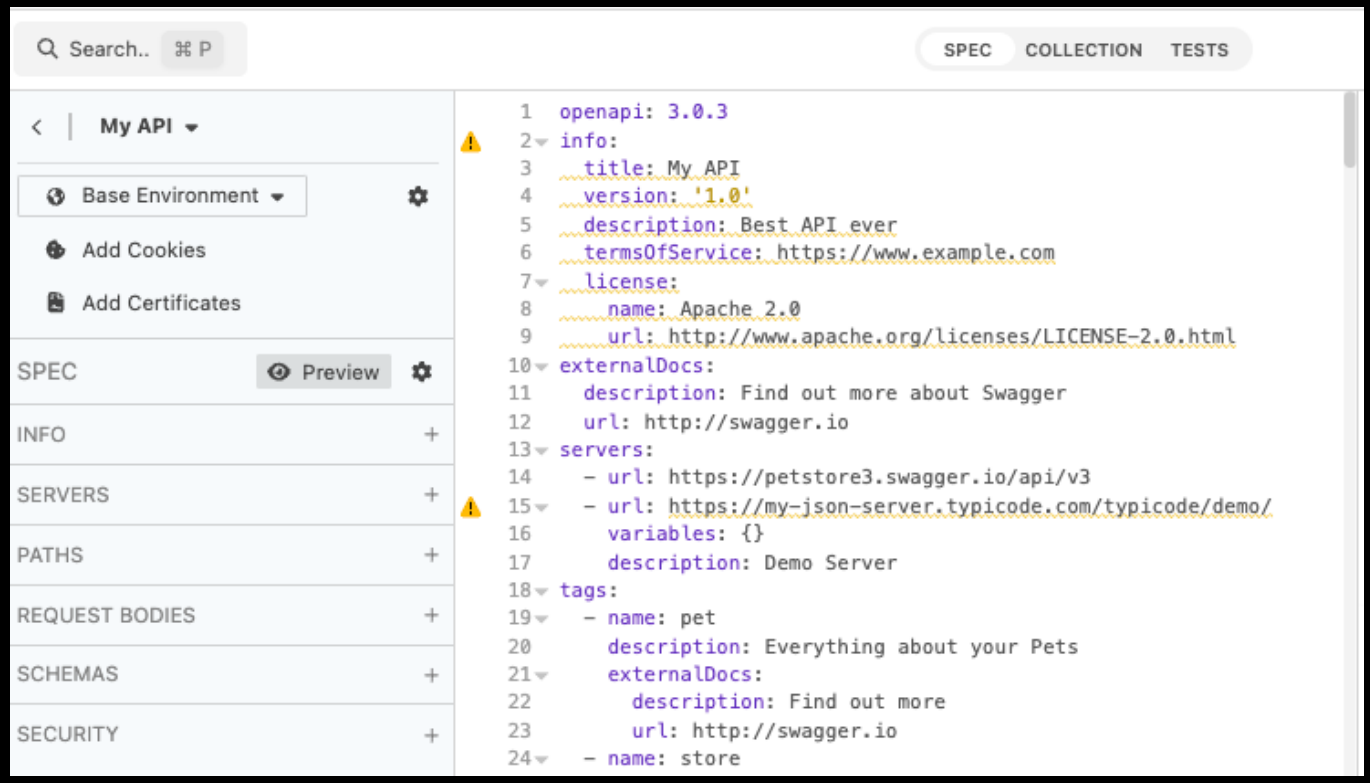




API Design: Code First

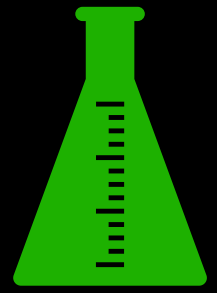
- **Paste your texts from the Swagger editor into the Insomnia editor**

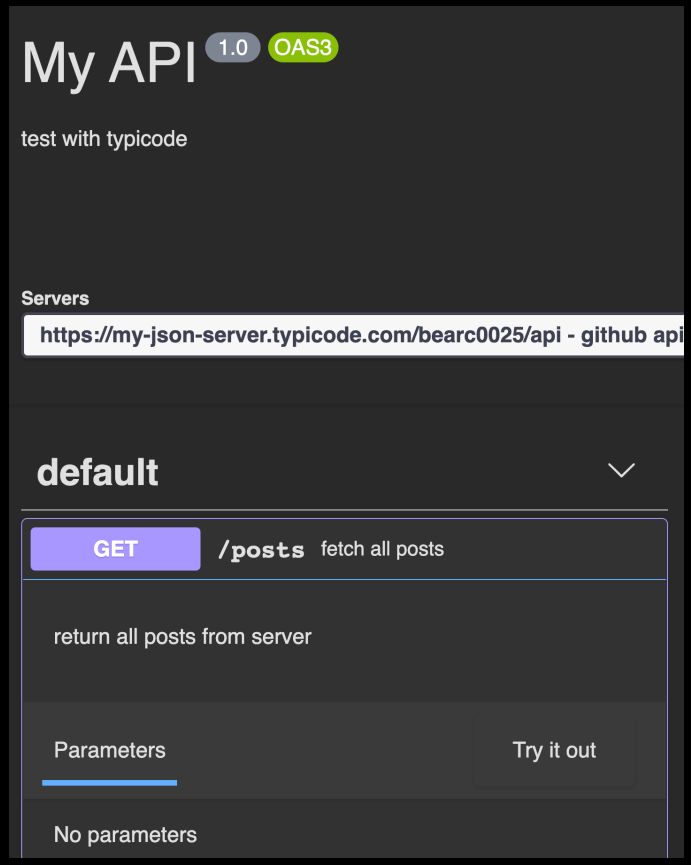




API Design: Code First

- **Test on the right**





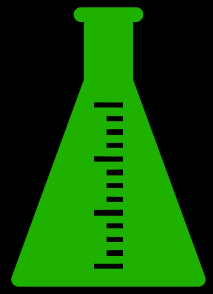
End of lab

API Design: Contracts

- **For more… PluralSight**

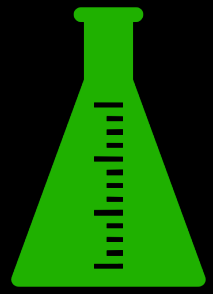
Designing RESTful Web APIs

Shawn Wildermuth



- **Lab: Local Typicode JSON Server**
  - **Local server**
  - **Same data as Typicode db.json file**
  - **Add server in OpenAPI design**

API Design: RESTful API Basics



- **Lab: github.com/typicode/json-server**
- **npm install -g json-server**
  Create db.json file with contents like previous test server
  https://my-json-server.typicode.com/typicode/demo/

Getting started

Install JSON Server

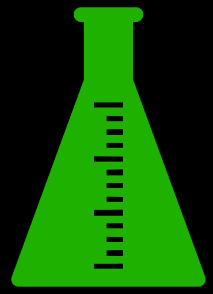
```
npm install -g json-server
```

Create a `db.json` file with some data

```
{
  "posts": [
    { "id": 1, "title": "json-server", "author": "typicode" }
  ],
  "comments": [
    { "id": 1, "body": "some comment", "postId": 1 }
  ],
```

```
{
  "posts": [
    { "id": 1, "title": "json-server", "author": "typicode" }
  "comments": [
  ],
    { "id": 1, "body": "some comment", "postId": 1 }
  ],
  "profile": { "name": "typicode" }
}
```

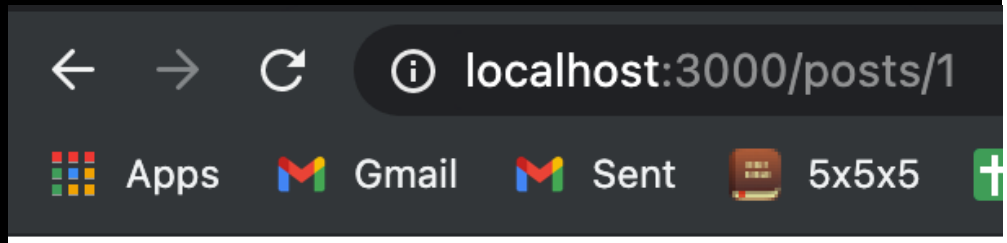
API Design: RESTful API Basics



- **Lab**
- **json-server --watch db.json**
  In the same folder as db.json

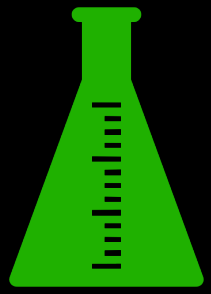
http://localhost:3000/posts
http://localhost:3000/posts/1

```
Bear-MBP:json-server bearc2020$ json-server --watch db.json

  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3000/posts
  http://localhost:3000/comments
  http://localhost:3000/reviews
  http://localhost:3000/profile

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```



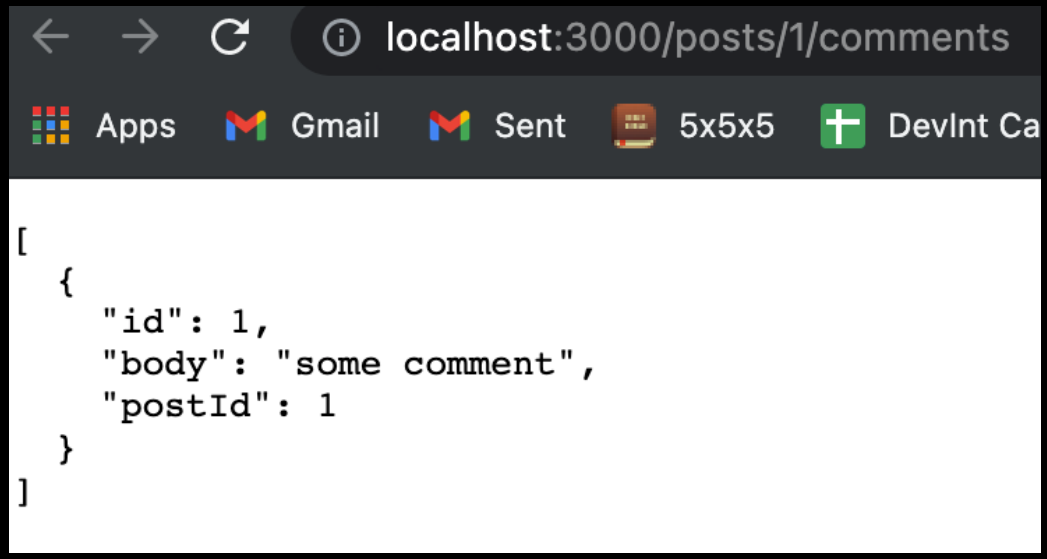

```
{
  "id": 1,
  "title": "json-server",
  "author": "typicode"
}
```

API Design: RESTful API Basics

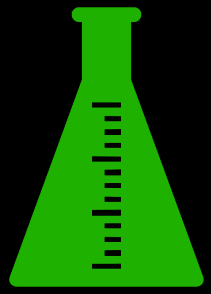


- **Lab**

http://localhost:3000/posts/1/comments

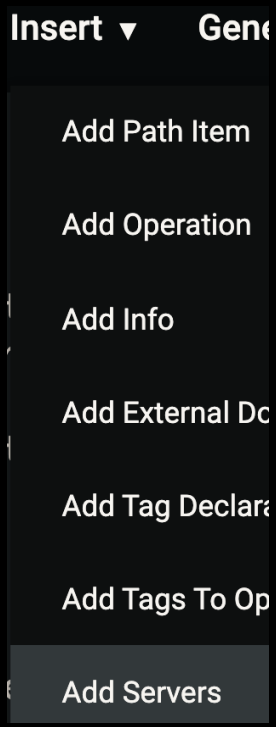


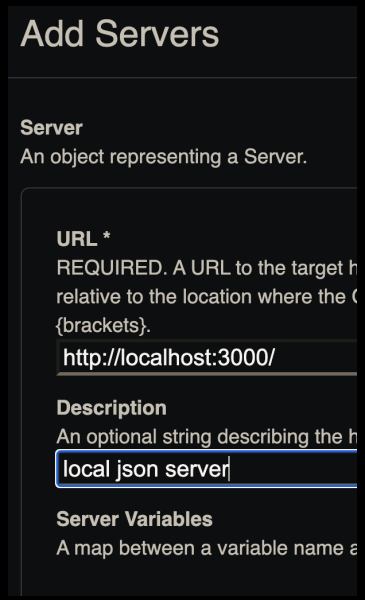
API Design: RESTful API Basics



- **Lab**

Add Server to Swagger editor



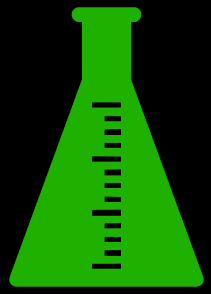


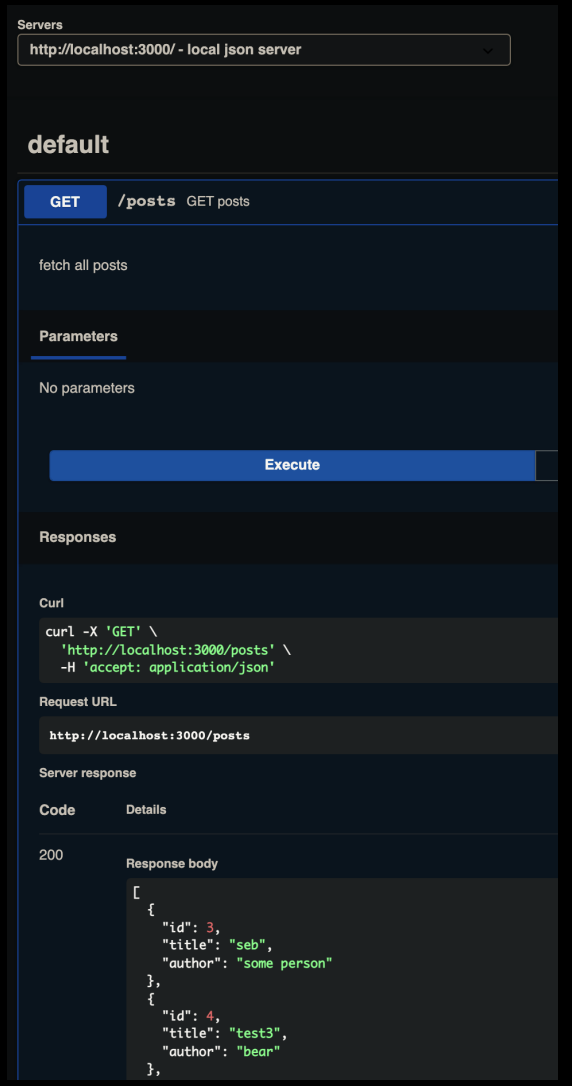

```yaml
servers:
  - url: https://my-json-server.typicode.com/bearc0025/api
    variables: {}
    description: Test server
  - url: http://localhost:3000/
    variables: {}
    description: local json server
```

API Design: RESTful API Basics

- **Lab**

Test against local server

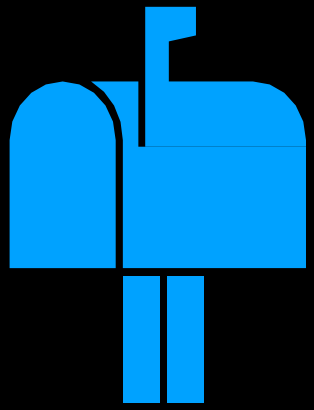




End of lab

API Design: RESTful API Basics

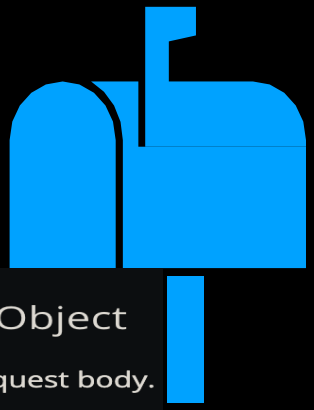
- **CRUD: Create POST HTTP Method**
- **Usually on base path e.g. /users HTTP**
- **Body with data: JSON**
  - **Server creates ID**
  - **Request Body without ID**



```
{
  "username": "bear",
  "score": 55,
  "active": true
}
```

API Design: RESTful API Basics

- https://swagger.io/specification/#parameter-object
- **Similar to Response/schema**
- **Not per-HTTP code**
- **Can be defined in Components**



Request Body Object

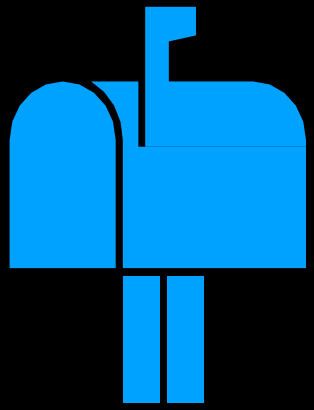
Describes a single request body.

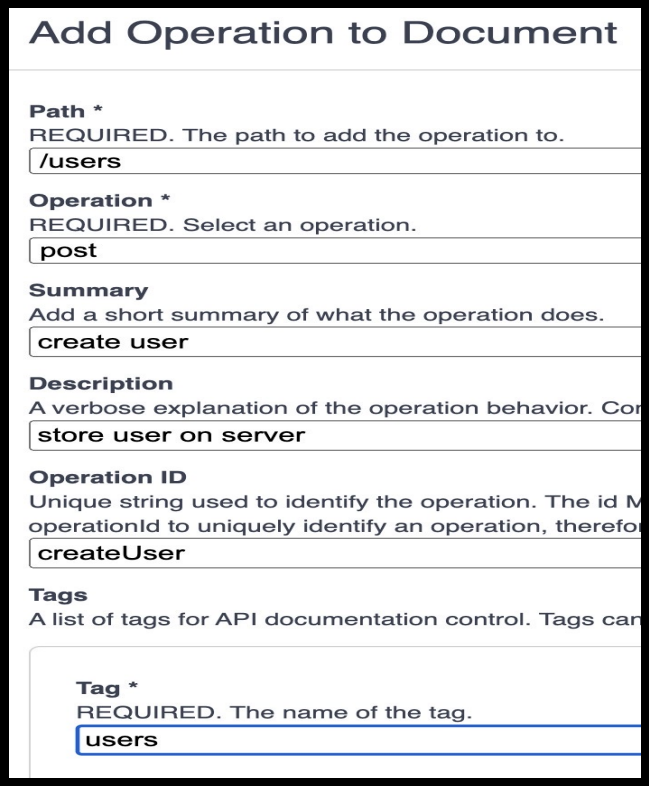
Fixed Fields

Field Name	Type
description	`string`
content	Map[`string`, Media Type Object]
required	`boolean`

API Design: RESTful API Basics

- **Lab: Add /users POST**
- **Add POST operation**
  - **/users**
  - **post**

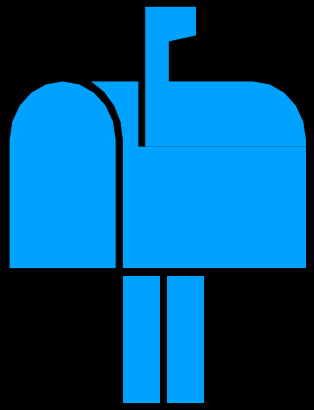





```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```

API Design: RESTful API Basics

- **Lab**
- **Without a Request Body section, the documentation doesn't specify what to send**
- **Add Request Body (typed out on next slide)**



```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```

API Design: RESTful API Basics

- Lab
- requestBody:
  - Starts section
- required: true
- content:
  - application/json
  - schema
    - object

```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          properties:
            username:
              type: string
              example: bear
            score:
              type: integer
              example: 55
            active:
              type: boolean
              example: true
```

API Design: RESTful API Basics

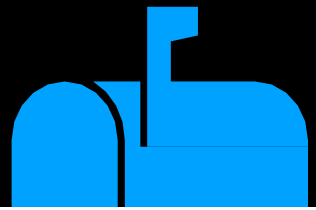
- Lab
- properties:
  - Object data
- <name> e.g., title
- type:
  - String, Integer, etc.
  - See https://swagger.io/specification/#data-types
- Can be required (otherwise, optional)

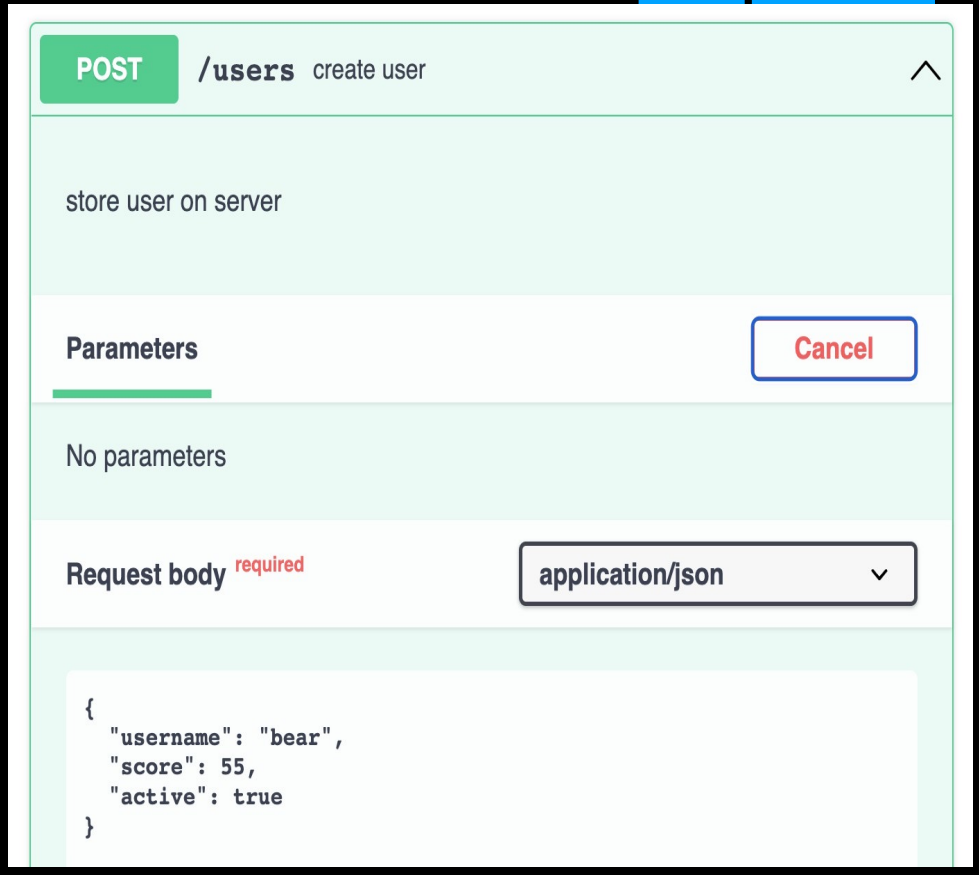
```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          properties:
            username:
              type: string
              example: bear
            score:
              type: integer
              example: 55
            active:
              type: boolean
              example: true
```

```yaml
type: object
properties:
  username:
    type: string
    example: bear
  score:
    type: integer
    example: 55
  active:
    type: boolean
    example: true
required:
  - username
```

API Design: RESTful API Basics

- **Lab**
- **Test in documentation**
- **Verify response**





- **End of lab**

API Design: RESTful API Basics

- **Lab: Add Request to POST**
- **Add PostNewUser to schemas:**
- **Add ref in post method requestBody:**

```yaml
requestBody:
  required: true
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/PostNewUser'
```

```yaml
PostNewUser:
  type: object
  properties:
    username:
      type: string
      example: bear
    score:
      type: integer
      example: 55
    active:
      type: boolean
      example: true
```

- **End of lab.**

API Design

- OpenAPI: Schemas
- Also supports:
- Wildcards contents: image/*
- anyOf, oneOf, allOf
- Files
- Multipart POSTs Form Data

```yaml
Dog:     # "Dog" is a value for the pet_type propert
  allOf: # Combines the main `Pet` schema with `Dog`
    - $ref: '#/components/schemas/Pet'
    - type: object
      # all other properties specific to a `Dog`
      properties:
        bark:
          type: boolean
        breed:
          type: string
          enum: [Dingo, Husky, Retriever, Shepherd]
```

```yaml
requestBody:
  description: A JSON object containing pet informa
  content:
    application/json:
      schema:
        oneOf:
          - $ref: '#/components/schemas/Cat'
          - $ref: '#/components/schemas/Dog'
          - $ref: '#/components/schemas/Hamster'
```

```yaml
examples:
  hamster:  # <--- example name
    summary: An example of a hamster
    value:
      # vv Actual payload goes here vv
      name: Ginger
      petType: hamster
```

API Design

- **OpenAPI: Schemas**
- **FullUser is the same as PostNewUser plus id**
  - **Define FullUser as PostNewUser and id**

```yaml
schemas:
  FullUser:
    type: object
    properties:
      id:
        type: string
        example: abc123
      username:
        type: string
        example: bear
      score:
        type: integer
        example: 55
      active:
        type: boolean
        example: true
  PostNewUser:
    type: object
    properties:
      username:
        type: string
        example: bear
      score:
        type: integer
        example: 55
      active:
        type: boolean
        example: true
```

API Design

- **OpenAPI: Schemas**
- **Set the schema to be ‘allOf:’**
- **Array (“-“)**
  - **Ref to FullUser**
  - **“type: object” like now but just id**
  - **End of lab**

```yaml
schemas:
  FullUser:
    allOf:
      - $ref: '#/components/schemas/PostNewUser'
      - type: object
        properties:
          id:
            type: string
            example: abc123
```

API Design

- **OpenAPI: Request Bodies**
- **Can be defined in Components**
- **Referenced in operations**

```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/PostNewUser'
```

```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    $ref: '#/components/requestBodies/UserPostBody'
```

```yaml
components:
  requestBodies:
    UserPostBody:
      description: new user request
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/PostNewUser'
```

- **End of lab**

API Design

- **OpenAPI: Responses**
- **Can be defined in Components**
- **Referenced in operations**

```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    $ref: '#/components/requestBodies/UserPostBody'
  responses:
    2XX:
      description: successful fetch
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/FullUser'
```

- **End of lab**

```yaml
post:
  summary: create user
  description: store user on server
  operationId: createUser
  requestBody:
    $ref: '#/components/requestBodies/UserPostBody'
  responses:
    2XX:
      $ref: '#/components/responses/UserRespBody'
    default:
      description: Default error sample response
  tags:
    - users
```

```yaml
components:
  responses:
    UserRespBody:
      description: user response
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/FullUser'
```

API Design

- **OpenAPI: Responses**
- **Arrays**
- **/users GET returns an array**
- **That can be another schema in another response use in /users GET**

```yaml
schemas:
  UserArray:
    type: array
    items:
      $ref: '#/components/schemas/FullUser'
```

```yaml
responses:
  UserArrayResponseBody:
    description: user array response
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/UserArray'
```

```yaml
/users:
  summary: user level operations
  description: create and read users
  get:
    summary: fetch users
    description: read users from server
    operationId: fetchUsers
    responses:
      2XX:
        $ref: '#/components/responses/UserArrayResponseBody'
      default:
        description: Default error sample response
```

- **End of Lab**

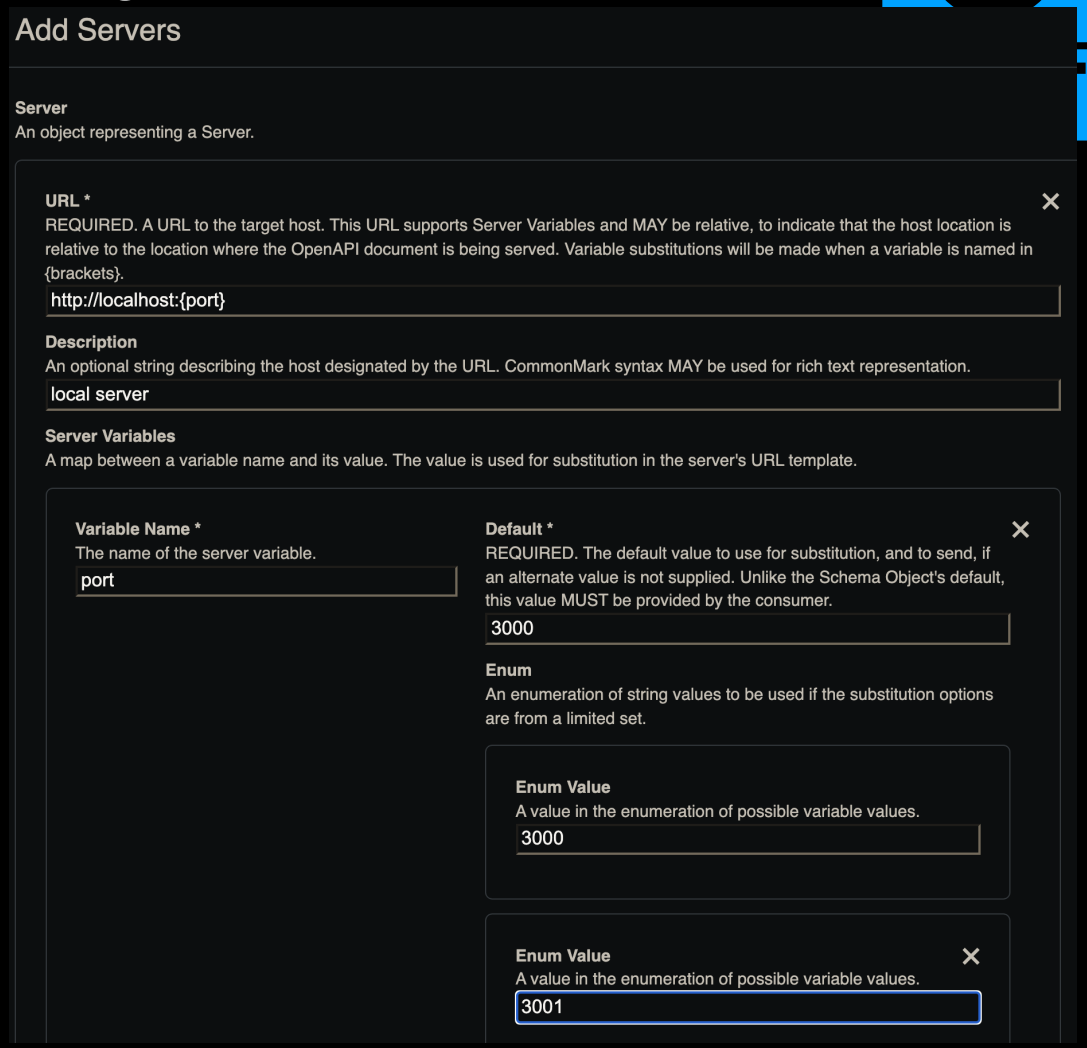
API Design

- **Components**
- **Reusable items**
  - **Request bodies, responses, schemas, parameters, etc.**
  - **Can become the bulk of your spec**
  - **Helps keep the paths/operations:**
    - **Concise**
    - **Description**
    - **Many references**

API Design

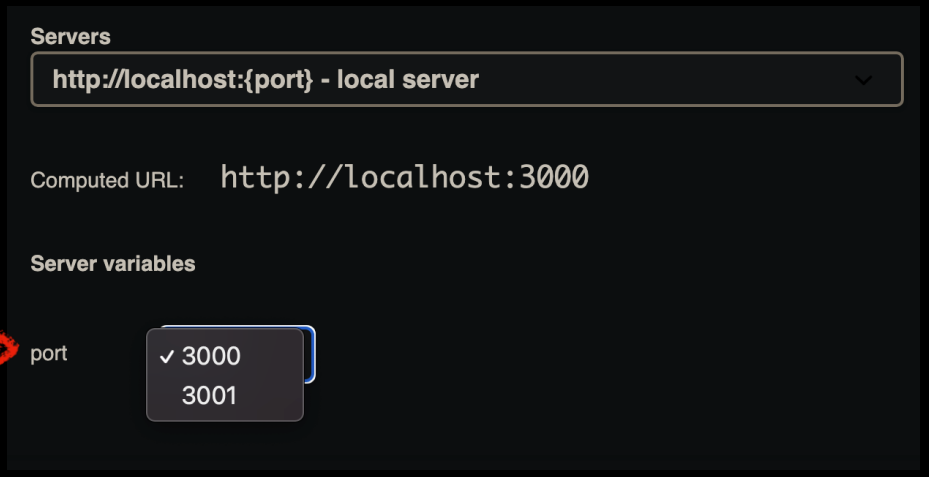
API Design

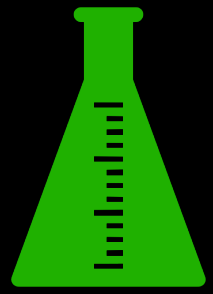
- **OpenAPI: Servers**
  - **Server Variables**
    - **Port**
    - **Version**
    - **Environment**
      - **Test, dev, etc.**



- **OpenAPI: Servers**
  - **Multiple URLs**
    - **e.g., scheme://host[:port][/path]**
  - **Variables**
    - **Client responsible for providing URL**

```yaml
- url: http://localhost:{port}
  variables:
    port:
      default: '3000'
      enum:
        - '3000'
        - '3001'
      description: ''
  description: local server
```





- **Lab: OpenAPI: Servers**

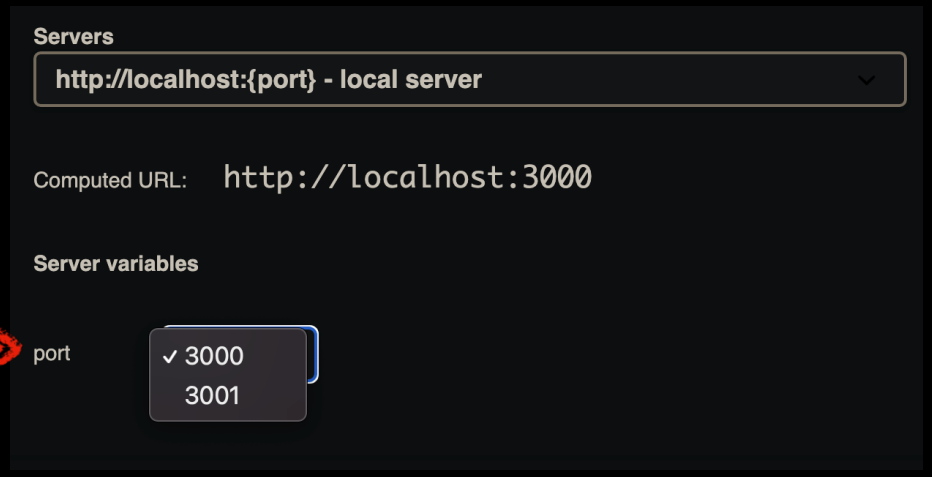
Multiple URLs

e.g., scheme://host[:port][/path]

Variables

Client responsible for providing URL

```
- url: http://localhost:{port}
  variables:
    port:
      default: '3000'
      enum:
        - '3000'
        - '3001'
      description: ''
  description: local server
```



- **OpenAPI: Paths and Parameters**

May be templated:

/users/{userId}

/department/{deptId}/employees

/employees/{empId}/department

Client responsible for providing values

parameters (array) describes variables

```yaml
parameters:
  - name: userId
    in: path
    required: true
    description: Parameter
    schema:
      type : integer
      format: int64
      minimum: 1
```

- **OpenAPI: Parameters**

in:
path - URL
query - Query String
header - request header
cookie - Cookie

```yaml
parameters:
  - name: userId
    in: path
    required: true
    description: Parameter
    schema:
      type : integer
      format: int64
      minimum: 1
```

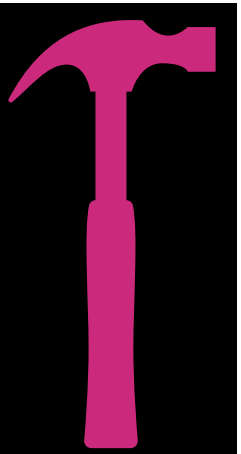
- **OpenAPI: Parameters**

path - URL

/user/{userId}

e.g., /user/223

```yaml
parameters:
  - name: userId
    in: path
    required: true
    description: Parameter
    schema:
      type : integer
      format: int64
      minimum: 1
```



- **Lab: /posts/{postId}**

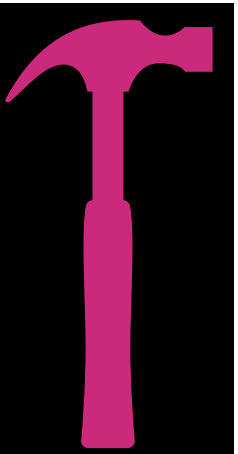
Add /posts/{postId}

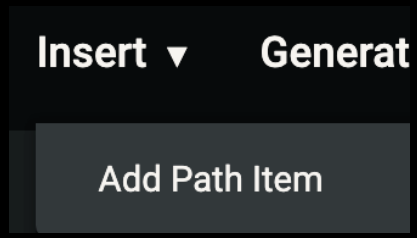
Add GET method

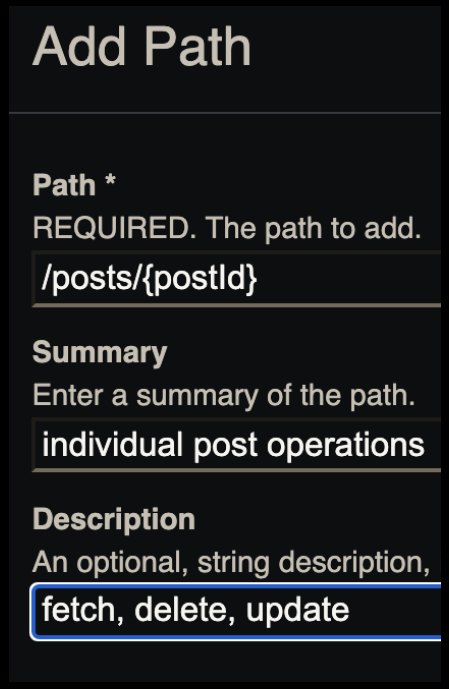
Include id parameter of type Integer

- **Lab: /posts/{postId}**

Add /posts/{postId}

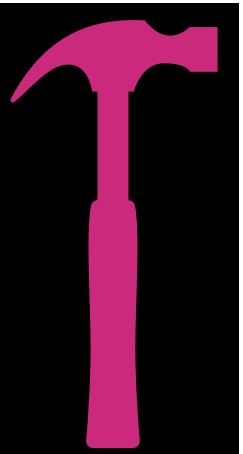


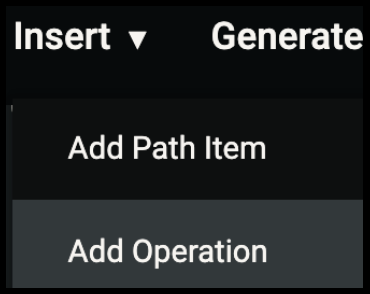


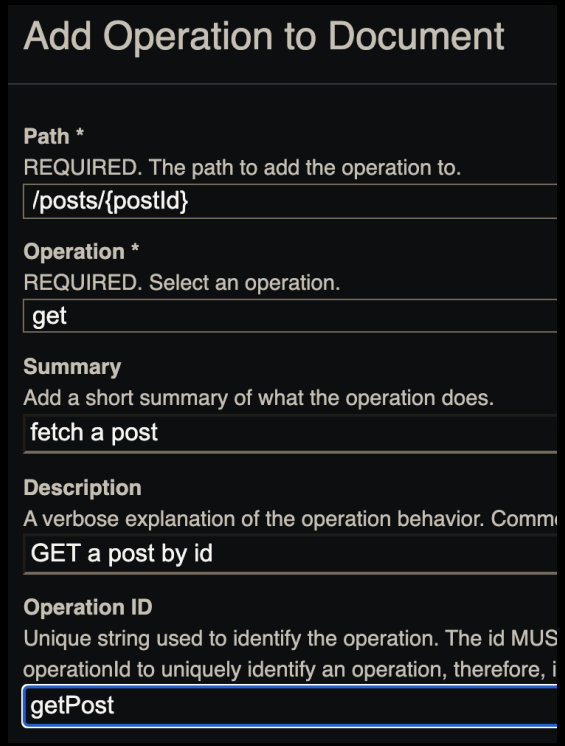



```
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
```

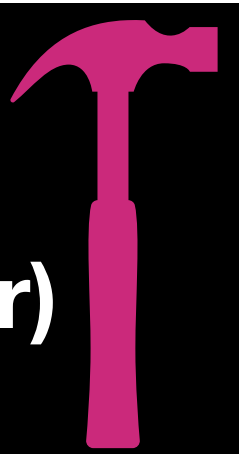
- **Lab: /posts/{postId}**
  Add GET Operation








```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  get:
    summary: fetch a post
    description: GET a post by id
    operationId: getPost
    responses:
      default:
        description: Default error sample response
```

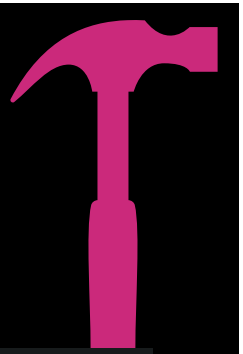


- **Lab: /posts/{postId}**
  Add GET parameters section (type in editor)

```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  get:
    summary: fetch a post
    description: GET a post by id
    operationId: getPost
    parameters:
      - in: path
        required: true
        name: postId
        schema:
          type: integer
          minimum: 0
          example: 5
```

- **OpenAPI: Parameters**
  query - Query String
  e.g., /posts?offset=20

```yaml
- in: query
  name: offset
  schema:
    type: integer
  description: The number of items to skip
```



- Lab

Add query string parameters

/posts path item

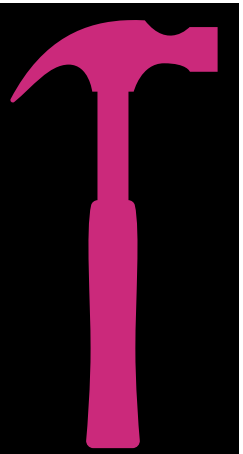
parameters section

Add offset

Add limit

Test Get method

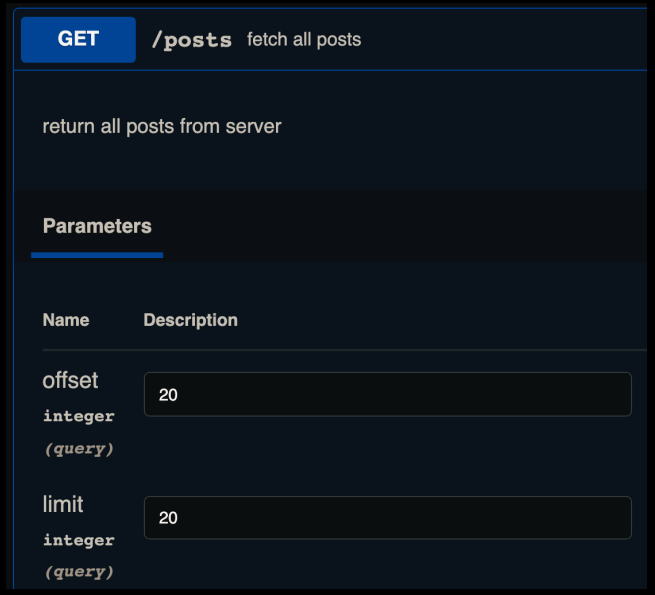
```yaml
paths:
  /posts:
    summary: posts level operations
    description: create new post, fetch all posts
    get:
      summary: fetch all posts
      description: returns all posts from server
      operationId: fetchPosts
      parameters:
        - name: offset
          in: query
          schema:
            type: integer
            minimum: 0
            example: 20
        - name: limit
          in: query
          schema:
            type: integer
            minimum: 1
            example: 20
```

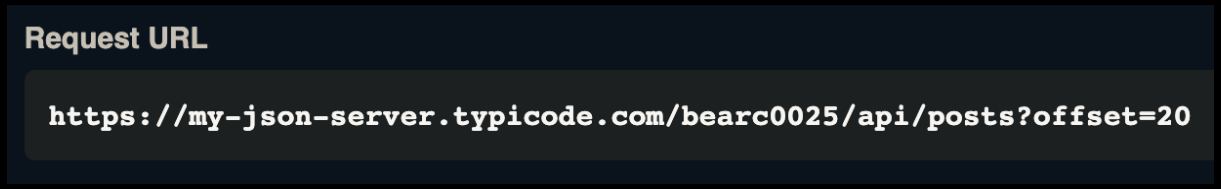


- Lab

Test Get method

End of lab





- **OpenAPI: Parameters**

  header - request header

  e.g., X-Request-ID: 77e1c83b-7bb0-437b-bc50-a7a58e5660ac

```yaml
paths:
  /ping:
    get:
      summary: Checks if the server is alive
      parameters:
        - in: header
          name: X-Request-ID
          schema:
            type: string
            format: uuid
          required: true
```

- **OpenAPI: Parameters**

cookie - Cookie

e.g., Cookie: debug=0;

csrftoken=BUSe35dohU3O1MZvDCUOJ

```yaml
parameters:
  - in: cookie
    name: debug
    schema:
      type: integer
      enum: [0, 1]
      default: 0
  - in: cookie
    name: csrftoken
    schema:
      type: string
```

- **OpenAPI: Parameters**
  - **Required: true/false**
  - **schema**
    - **Used for primitive types**
    - **Arrays**
    - **Serialized (to string) objects**
  - **content for more complex serialization**
    - **See https://swagger.io/docs/specification/serialization/**

```yaml
parameters:
  - name: userId
    in: path
    required: true
    description: Parameter
    schema:
      type : integer
      format: int64
      minimum: 1
```

- **OpenAPI: Parameters**
  - **May have default values**
    - **Not needed for required values**
  - **May have min/max**

```yaml
parameters:
  - in: cookie
    name: debug
    schema:
      type: integer
      enum: [0, 1]
      default: 0
  - in: cookie
    name: csrftoken
    schema:
      type: string
```

```yaml
parameters:
  - name: userId
    in: path
    required: true
    description: Parameter
    schema:
      type : integer
      format: int64
      minimum: 1
```

- **OpenAPI: Parameters**

May have an example(s)

May be an enum

```yaml
parameters:
  - in: query
    name: limit
    schema:
      type: integer
      minimum: 1
    example: 20
```

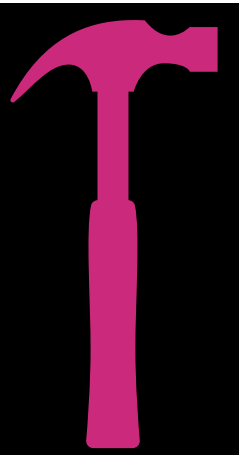
```yaml
parameters:
  - in: query
    name: status
    schema:
      type: string
      enum:
        - available
        - pending
        - sold
```

- **OpenAPI: Parameters**

Parameters shared by multiple methods may be defined on the path level.

e.g., Used by GET and DELETE

```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - in: path
      required: true
      name: postId
      schema:
        type: integer
        minimum: 0
        example: 5
```



- **Lab: /posts/{postId}**

Put parameters section on path level

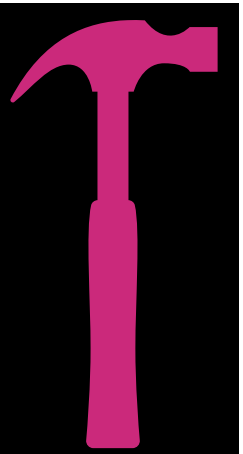
Applies to all operations for this path

```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - in: path
      required: true
      name: postId
      schema:
        type: integer
        minimum: 0
        example: 5
```

- **OpenAPI: Parameters**
  May be in components
  Referenced with $ref

```yaml
components:
  parameters:
    PostId:
      in: path
      required: true
      name: postId
      schema:
        type: integer
        minimum: 0
        example: 5
```

```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - $ref: '#/components/parameters/PostId'
```



- **Lab: /posts/{postId}**
  Put parameter in parameters section of components
  Refer from operation /posts/{postId}

```yaml
components:
  parameters:
    PostId:
      in: path
      required: true
      name: postId
      schema:
        type: integer
        minimum: 0
        example: 5
```

```yaml
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - $ref: '#/components/parameters/PostId'
```

- **OpenAPI: Parameters - Serialization**

For transmitting data structures/objects

Keywords:

style - how values are delimited

Various by parameter location (e.g., path)

explode (boolean)- specifies if values should be separate parameters (e.g., arrays)

API Design

- **Path Parameters**

Styles:

simple (default)

label

matrix

Explode

true

false (default)

API Design

- **Styles: simple - primitive**
  Looks like /rates/{id}
  Parameter as we've seen it before
  Required
  Schema type: integer

```
r.typicode.com/bearc0025/api/rates/1
```

```yaml
/rates/{id}:
  summary: review stuff
  description: operations on item
  get:
    summary: fetch data
    description: returns data from db
    parameters:
      - in: path
        required: true
        name: id
        schema:
          type: integer
```

API Design

- Styles: simple - array
  Array of integers
  explode true: /rates/{id*}
  explode false: /rates/{id}
    URL the same

```
on-server.typicode.com/bearc0025/api/rates/1,2,4
```

```yaml
/rates/{id*}:
  summary: review stuff
  description: operations on item
  get:
    summary: fetch data
    description: returns data from db
    parameters:
      - in: path
        style: simple
        explode: false
        required: true
        name: id*
        schema:
          type: array
          items:
            type: integer
```

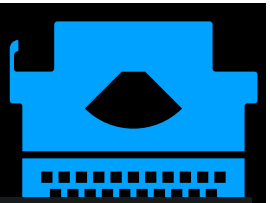
API Design

- **Styles: simple - object**
  Object
  explode: false
    Comma separated

```
-json-
icode.com/bearc0025/api/rates/id,1,text,this%20post
```

```yaml
/rates/{item}:
  summary: review stuff
  description: operations on item
  get:
    summary: fetch data
    description: returns data from db
    parameters:
      - in: path
        style: simple
        explode: false
        required: true
        name: item
        schema:
          type: object
          properties:
            id:
              type: integer
              example: 1
            text:
              type: string
              example: this post
```

API Design



- **Styles: simple - object**
  Object
  explode: true
  Key/Value pairs

```
.com/bearc0025/api/rates/id=1,text=this%20post
```

```yaml
/rates/{item*}:
  summary: review stuff
  description: operations on item
  get:
    summary: fetch data
    description: returns data from db
    parameters:
      - in: path
        style: simple
        explode: true
        required: true
        name: item*
        schema:
          type: object
          properties:
            id:
              type: integer
              example: 1
            text:
              type: string
              example: this post
```

API Design

- **Styles: label - primitive/array**

Dot-prefixed (aka Label Expansion)

{.item}

Request URL

https://my-json-server.typicode.com/bearc0025/api/rates/.1

{.item*} for array

URL

://my-json-server.typicode.com/bearc0025/api/rates/.1.2

```yaml
/rates/{.item}:
  summary: review stuff
  description: operations on ite
  get:
    summary: fetch data
    description: returns data fr
    parameters:
      - in: path
        style: label
        explode: false
        required: true
        name: .item
        schema:
          type: integer
```

- **Styles: label - object**

explode: false

```
URL

//my-json-
.typicode.com/bearc0025/api/rates/.id.1.text.this%20post
```

explode: true

```

y-json-
picode.com/bearc0025/api/rates/.id=1.text=this%20post
```

```yaml
parameters:
  - in: path
    style: label
    explode: false
    required: true
    name: .item
    schema:
      type: object
      properties:
        id:
          type: integer
          example: 1
        text:
          type: string
          example: this post
```

API Design

- **Styles: matrix**

Primitive

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates/;item=1
```

Array - Explode

False

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates/;item=1,2
```

True

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates/;item=1;item=2
```

- Styles: matrix

Object

Explode false

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates/;item=id,1,text,this%20post
```

Explode true

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates/;id=1;text=this%20post
```

- **Query Parameters**

Styles

form

spaceDelimited

pipeDelimited

deepObject

- Query Parameter Styles

form

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates?item=1
```

```yaml
/rates:
  summary: review stuff
  description: operations on item
  get:
    summary: fetch data
    description: returns data from db
    parameters:
      - in: query
        name: item
        style: form
        explode: true
        schema:
          type: integer
```

Array

Explode false

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates?item=1,2
```

Explode true

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates?item=1&item=2
```

- **Query Parameter Styles**

spaceDelimited - (explode false)

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates?item=1%202
```

pipeDelimited - (explode false)

Request URL

```
https://my-json-server.typicode.com/bearc0025/api/rates?item=1|2
```

For explode = true, looks like form

- **Query Parameter Styles**

deepObject (Object)

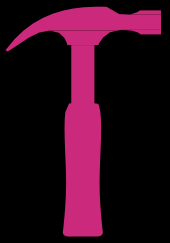
%5B = [

%5D =]

?item[id]=1&item[text]=this post

Request URL

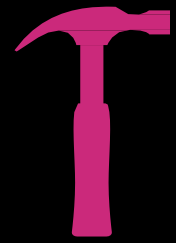
```
https://my-json-server.typicode.com/bearc0025/api/rates?
item%5Bid%5D=1&item%5Btext%5D=this%20post
```



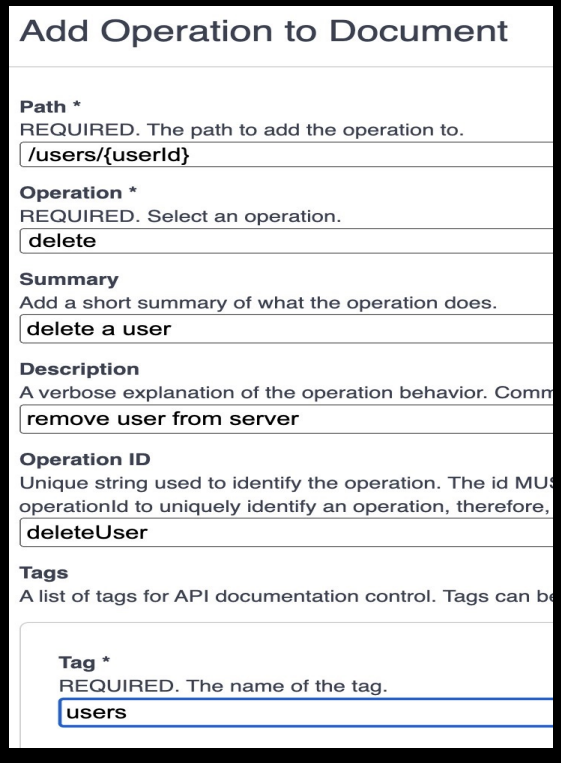
- **Lab: Swagger DELETE with ID**
- **Move {id} parameter to components**
  - **(If haven't already)**

```
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - $ref: '#/components/parameters/PostId'
```

API Design

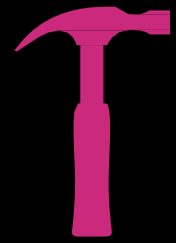


- **Lab: Swagger DELETE with ID**
- **Add DELETE operation/method to /users/{userId} (Insert -> Add Operation)**
- **Test**




```
delete:
  summary: delete a user
  description: remove user from server
  operationId: deleteUser
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```

- **End of Lab**



- **API Design**
  - **Lab: Swagger PATCH with ID**
  - **Add patch operation to /users/{userId} Very similar to post on /users**
  - **NOTE: Be sure to add the requestBody**
  - **Test**

Path *

REQUIRED. The path to add the operation to.

/users/{userId}

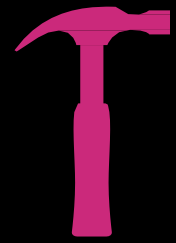
Operation *

REQUIRED. Select an operation.

patch

```yaml
patch:
  summary: update a user
  description: store changes on server
  operationId: updateUser
  requestBody:
    $ref: '#/components/requestBodies/UserPostBody'
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```

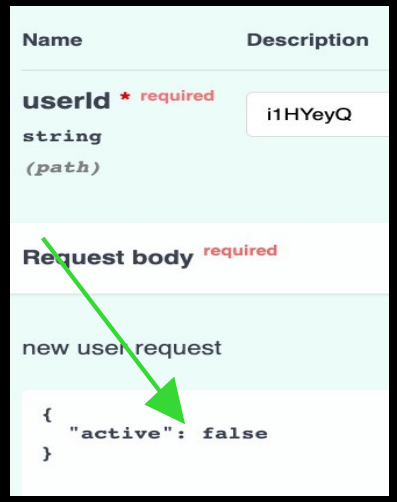
API Design



- **API Design**
- **Lab: Swagger PATCH with ID**
- **Update a user and verify the changes made are stored and existing values are otherwise not changed**

Response body

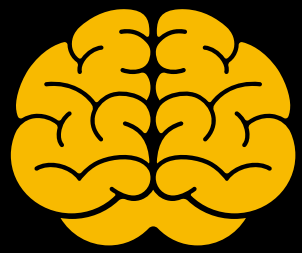
```
{
  "username": "bear",
  "score": 55,
  "active": true,
  "id": "i1HYeyQ"
}
```



Response body

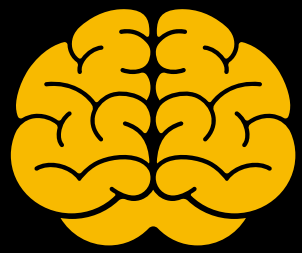
```
{
  "username": "bear",
  "score": 55,
  "active": false,
  "id": "i1HYeyQ"
}
```

API Design: Design First



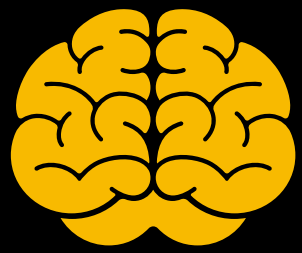
API Design: Design First

- Best for stability/maintainability
- Requires commitment
  - Keep up to date later
- Ideally know all data/endpoints
- Mocks/test
- Client feedback is earlier
- Benefits from code generators
  - Code can be questionable

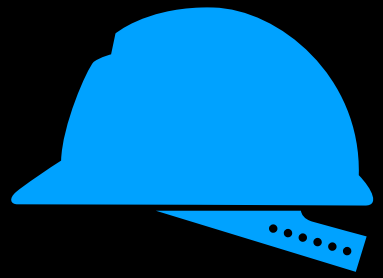


API Design: Design First

- **What could go wrong?**
  - **Reality - backend/db isn't as expected when implemented**
  - **Client needs aren't served**
  - **Large/many queries for little data**
- **Good for exposing data to unknown clients and uses**

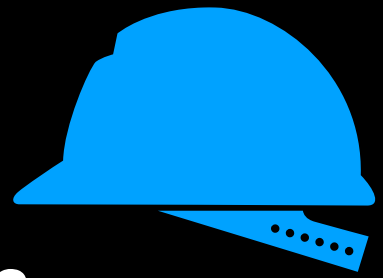


API Design: Code First



API Design: Code First

- **Best for speed**
- **Flexible**
- **Doesn’t solve problems that don’t exist**
- **Uncovers problems without design**
- **Documentation suffers**
- **Harder to maintain/communicate**

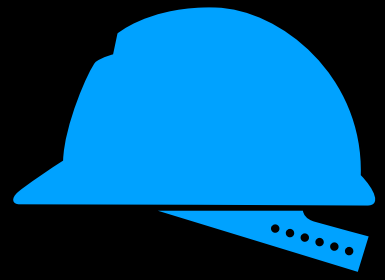


API Design: Code First

What could go wrong?

- **Code doesn’t map to endpoints, parameters**
- **Doc/communication fails**
- **API solidification causes delays**

Good for in-house data and quick updates internally (especially for small, tight teams)



API Design: Automated Testing

Benefits

- **Agnostic**
- **Isolated**
- **Repeatable**
- **Fast**

API Design: Automated Testing

- **Focus On What You're Testing**
  - **Functionality = Unit Tests**
  - **GUI = UI Tests**
  - **API = Endpoints, Format, etc.**
- **Errors**
  - **Proper codes**
  - **Messages**
  - **Graceful failures**

API Design: Automated Testing

- **Tools**
  - **Mock servers**
  - **Postman, Insomnia, Paw, etc.**
- **Fully Fleshed out OpenAPI**
  - **Endpoints & Methods**
  - **Request Bodies**
  - **Responses**
  - **Components/Model**
- **SDK Generation if applicable**

- **Insomnia**

"Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework."

API Design: Automated Testing

- **Insomnia**

https://www.chaijs.com/api/bdd/

BDD

The BDD styles are `expect` and `should`. Both use the same chainable language to construct assertions, but they differ in the way an assertion is initially constructed. Check out the Style Guide for a comparison.

API Reference

Language Chains

The following are provided as chainable getters to improve the readability of your assertions.

Chains

- to
- be
- been
- is
- that
- which
- and
- has
- have
- with
- at
- of
- same
- but
- does
- still
- also

.lengthOf(n[, msg])

- @param *{Number}* n
- @param *{String}* msg _optional_

Asserts that the target's `length` or `size` is equal to the given number `n`.

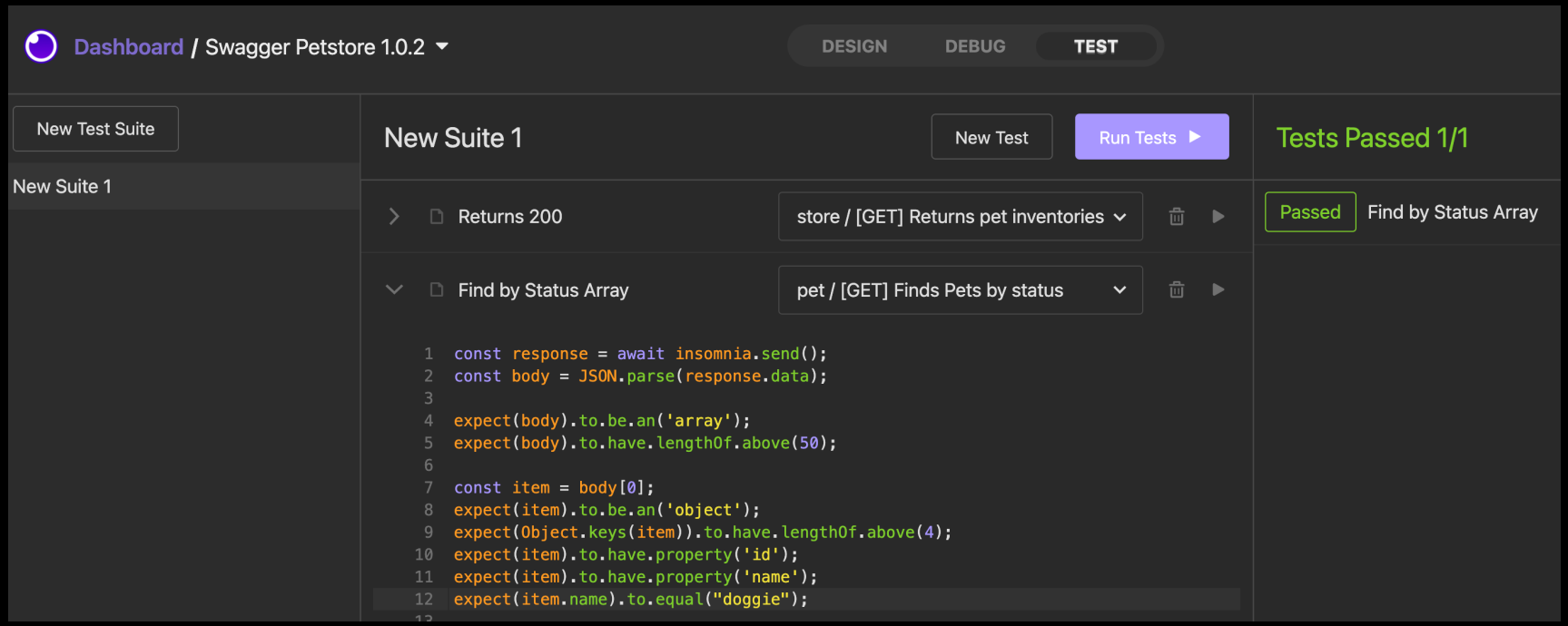
```
expect([1, 2, 3]).to.have.lengthOf(3);
expect('foo').to.have.lengthOf(3);
expect(new Set([1, 2, 3])).to.have.lengthOf(3);
expect(new Map([['a', 1], ['b', 2], ['c', 3]])).to.have.lengthOf(3);
```

API Design: Automated Testing

- **Insomnia**

Create Test Suite

Add tests and 'expects'



- **Insomnia**

Create Test Suite

Add tests and 'expects'

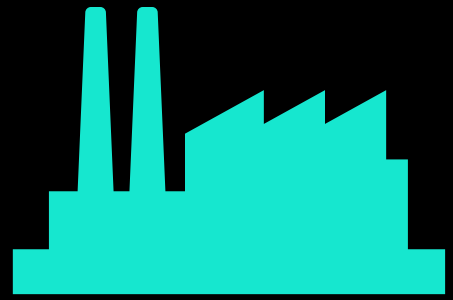
```
const response1 = await insomnia.send();

expect(response1.status).to.equal(200, "not 200");

const body = JSON.parse(response1.data);

expect(body).to.be.an('array');
expect(body).to.have.lengthOf.greaterThan(0);
expect(body[0]).to.have.property('id');
expect(body[0]).to.have.property('username');
```

API Design: Automated Testing

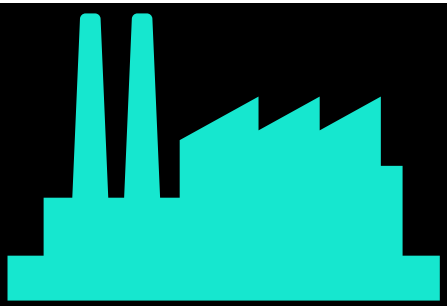
- **Insomnia and other tools can…**
  - **Lint your spec**
    - **Look for and trap small issues with big effects in syntax and similar**
  - **Run Tests**
    - **CI/CD on checkin, automated**
  - **Define config**
  - **Generate Code, Documentation**

API Design: Legacy Systems

What is the Legacy system?
How to map to REST nouns
  What are the data?
  Tables? Columns? Join?
  e.g., Report generation -> /report
Focus on API, state (not functionality)
API might be a facade
Treat the client and server as equals

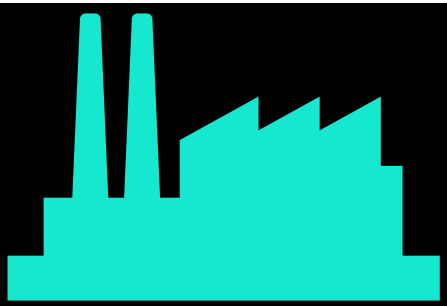


API Design: Legacy Systems



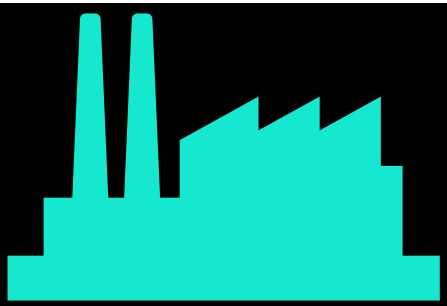
- Is the Legacy system based on database tables?
  - Nouns may be based on tables
  - Verbs convert straight from CRUD
  - Functionality can be minimal
  - Authentication always a concern

API Design: Legacy Systems



- Is the Legacy system based on functionality?
- Nouns may be based on results
- May only have GET for some endpoints (e.g., /reports)
- Authentication always a concern

API Design: Legacy Systems



Is the Legacy system based on functionality?

May use route aliases for grouping or common requests

e.g., /reports/today, /reports/yest

API Design: Versioning

API Design: Versioning

- **Versioning**
  - **In URL**
    - **Easy, clear**
    - **Requires URL changes for clients**

…example.com/api/v2/reports/34

API Design: Versioning

- **In Header**

  Might be better for HATEOAS

  Limits clients - requires header

  Accept Header:

  e.g., Accept:

  application/vnd.example+json;version=1.0

  Custom Header:

  e.g., Accept-version: v1

API Design: Versioning

- **Both**
  Major in URL - defaults to latest minor
  Allows for updates

API Design: Versioning

- **Backward Compatible**

  Old versions -> old code

  Or new code's version of functionality

  Behind the scenes implementation

API Design: Versioning

- **Deprecated version/endpoint**
  Freeze code
    Reliable
    Easier to isolate and update/fix

API Design: Security by Design

API Design: Security by Design

- **Consider security as 1st class entity**

SSL

HTTP Basic Auth (header)

API keys, secrets

Dangerous, Changes

http://www.omdbapi.com/?apikey=<api key>&t=memento

OAuth

Authentication, Authorization

JWT (Tokens)

ID

Access

```
GET /api/rest/contracts/status?contractId=42 HTTP/1.1
Authorization: Basic ZXh0ZXJuYWwtc2VydmljZW5hbWU6YXBpa2V5
```

API Design: Security by Design

- **Authentication vs Authorization**
  Client needs to be authenticated
    e.g., username + password
  Results in authorization
    e.g., token
    Has certain permissions (and not others)

API Design: JSON Web Tokens

- **JWT**

https://jwt.io/introduction

<header>.<payload>.<signature>

https://openid.net/developers/jwt/

Authentication/Authorization

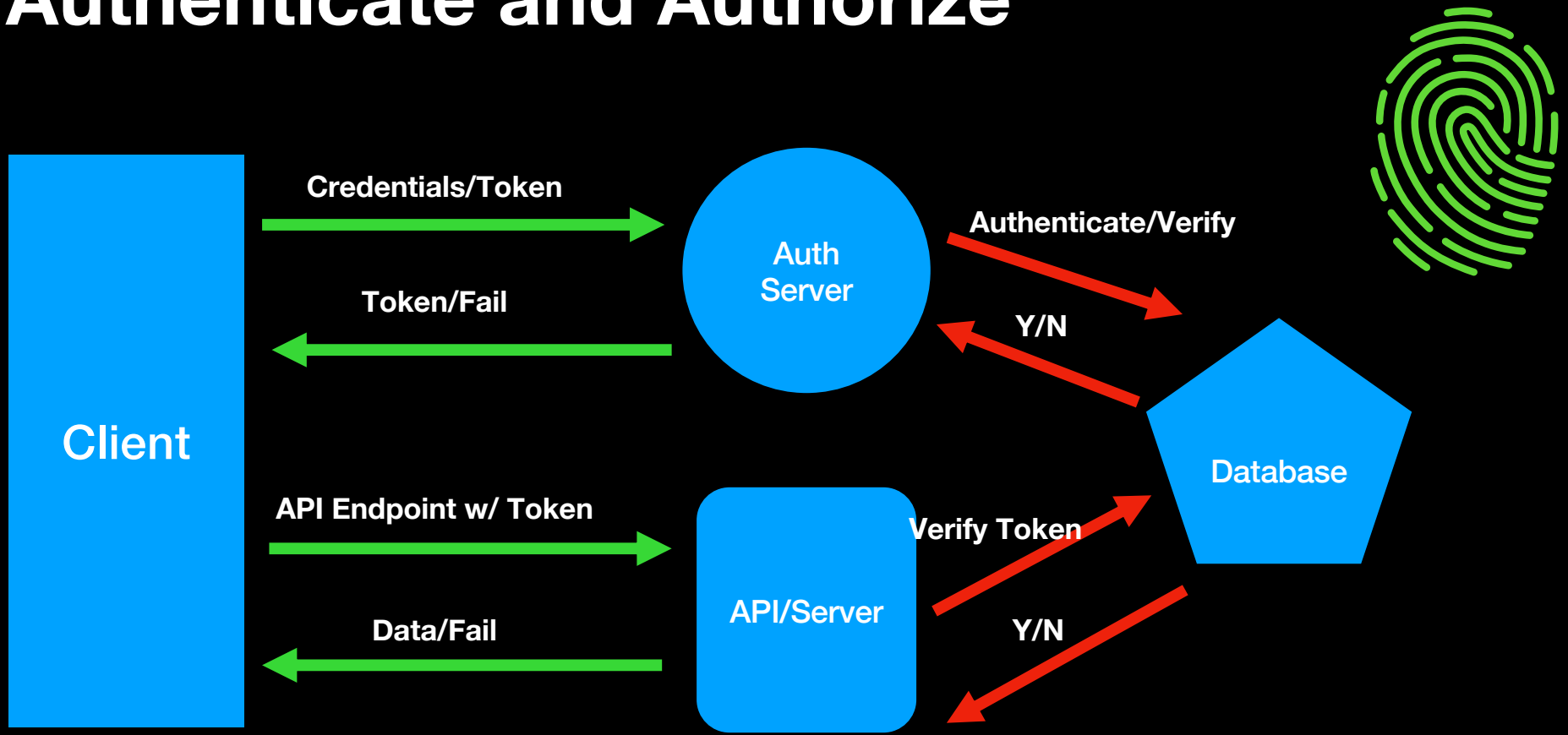
Access - http header

Expiration - refresh

SDKs often handle bulk

Authorize by user/role

- **Authenticate and Authorize**



API Design: Security by Design

- **securitySchemes in components for reuse**

```yaml
components:
  securitySchemes:

    BasicAuth:
      type: http
      scheme: basic

    BearerAuth:
      type: http
      scheme: bearer

    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key

    OpenID:
      type: openIdConnect
      openIdConnectUrl: https://example.com/.well-known/openid-config

    OAuth2:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: https://example.com/oauth/authorize
          tokenUrl: https://example.com/oauth/token
          scopes:
            read: Grants read access
            write: Grants write access
            admin: Grants access to admin operations
```

API Design: Security by Design

- **Example: Cognito**

```yaml
components:
  securitySchemes:
    MyAppAuth:
      type: "apiKey"
      name: "token"
      in: "header"
      x-amazon-apigateway-authtype:
        "cognito_user_pools"
```

```yaml
paths:
  /v2/alarm:
    get:
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/sc
      security:
      - MyAppAuth: []
```

API Design: Security

- **Security**
- **Security Scheme Object securitySchemes in Components**
- **Supported schemes:**
- **HTTP authentication**
- **API key (header, cookie, query)**
- **Mutual TLS (server & client certs)**
- **OAuth2 (password, client credentials, authorization code)**

API Design: Security

- **Fields**
- **type : string**
- **Values: apiKey, http, mutualTLS, oauth2, openIdConnect**
- **description : string**
- **Text description**

- **Each type has their own fields…**

API Design: Security

```
type: apiKey
name: api_key
in: header
```

- **apiKey fields:**
- **name**
- **The name of the header, cookie or query string parameter where the API key is passed.**
- **in**
- **Where the API key is passed: query, header, cookie**

API Design: Security

```
type: http
scheme: bearer
bearerFormat: JWT
```

- **http fields:**
- **scheme**
- **HTTP authorization used**
- **Values should be in IANA registry**
- **e.g., Basic, Bearer, etc.**
- **bearerFormat (for http bearer scheme)**
- **Hint for token format (e.g., JWT)**

API Design: Security

```
type: http
scheme: bearer
bearerFormat: JWT
```

- **http fields:**
- **scheme**
- **HTTP authorization used**
- **Values should be in IANA registry**
- **e.g., Basic, Bearer, etc.**
- **bearerFormat (for http bearer scheme)**
- **Hint for token format (e.g., JWT)**

API Design: Security

- **oauth2 fields:**
- **flows (Object)**
- **implicit**
- **password**
- **clientCredentials**
- **authorizationCode**

- **The above 4 fields are flow objects…**

API Design: Security

- **flow fields:**
- **authorizationURL (implicit and authorizationCode)**
- **tokenURL (password, clientCredentials, authorizationCode)**
- **refreshURL (all)**
- **scopes (all) - [string,string]**
- **e.g., "write:pets": "modify…"**
- **Example...**

API Design: Security

```
type: oauth2
flows:
  implicit:
    authorizationUrl: https://example.com/api/oauth/dialog
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
  authorizationCode:
    authorizationUrl: https://example.com/api/oauth/dialog
    tokenUrl: https://example.com/api/oauth/token
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
```

API Design: Security

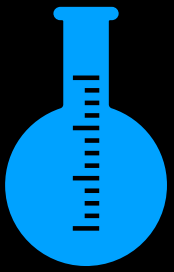
```
securitySchemes:
  api_key:
    type: apiKey
    name: api_key
    in: header
  petstore_auth:
    type: oauth2
    flows:
      implicit:
        authorizationUrl: http://example.org/api/oauth/dialog
        scopes:
          write:pets: modify pets in your account
          read:pets: read your pets
```

API Design: Security

- Security Requirement for Operation
- List required security schemes
  - If multiple, all must be satisfied
  - Refs Security Schemes in Components
  - For oauth2 and openIdConnect
    - May include required scopes
  - For others
    - May include role names

```
security:
- petstore_auth:
  - write:pets
  - read:pets
```

```
security:
- MyAppAuth: []
```



- **Principles**

Types of testing related to API:

UI

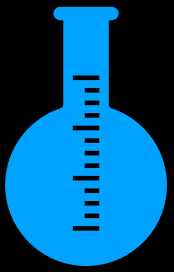
Functional

Security

Performance/Load

Error

Validation - end-to-end



- **Principles**

Types of testing related to API:

UI

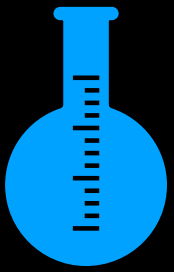
Creating Requests

Data format

Receiving Responses

May be client (not yours)

May be SDK



- **Principles**

Types of testing related to API:

Functional

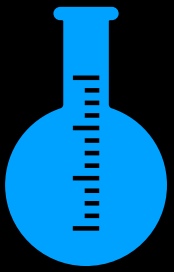
Back end

Code logic

Request Processing

Data Gathering, Formatting

Response Creation



- **Principles**

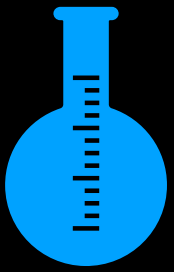
Types of testing related to API:

Security

Authentication

Authorization

Access Prevention



- **Principles**

Types of testing related to API:

Performance/Load

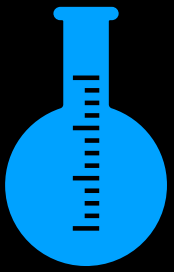
Process Speed

Scalability

Cost

DOS Security

Response Time



- **Principles**

Types of testing related to API:

Error

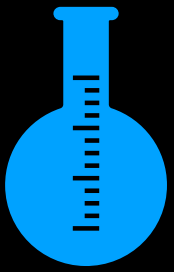
Fail Gracefully

Appropriate Status/Error Codes

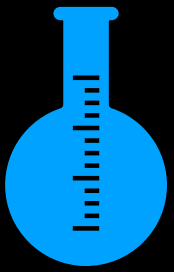
Messages

No Secrets

Logging

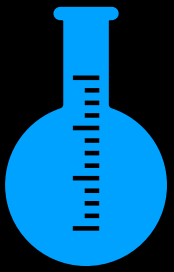


- **Best Practices**
  - **Use Realistic Data**
    - **Mock Data**
    - **Real Data**
    - **All Sizes**
    - **Include Errors**



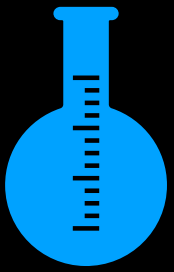
- **Best Practices**
  Test Success and Fail
  Don’t just test that it works
  Wild Testing

- **Best Practices**
  Drive Tests with Data
    Map test data to outcomes
    Repeatable
    Easily Extendable
    Fail Conditions Added



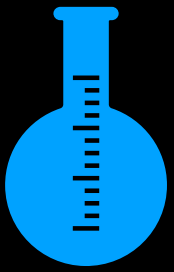
```
(100, 0.5)
(200, 0.75)
(300, 0.85)
(0, 0.0)
```

API Automated Testing



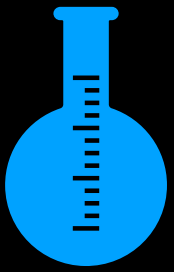
- **Best Practices**
  Logging/Analytics
  Store API traffic
  Analyze for changes
  Helps with bug research

API Automated Testing



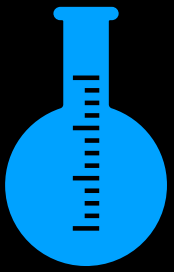
- **Best Practices**
  - **Performance Minded**
    - **Functional Tests can be reused**
      - **Performance**
      - **Scalability**
      - **Security**

API Automated Testing



- **Best Practices**
  - **Automation**
    - **Time $aving**
    - **Reliable/Repeatable**
    - **Agnostic to other software/processes**
    - **Right tool for the right job**
      - **Insomnia, Postman, Stoplight, etc.**
      - **CI/CD**

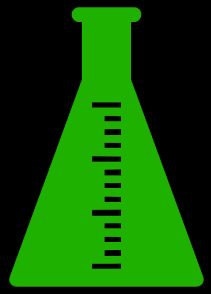
API Automated Testing

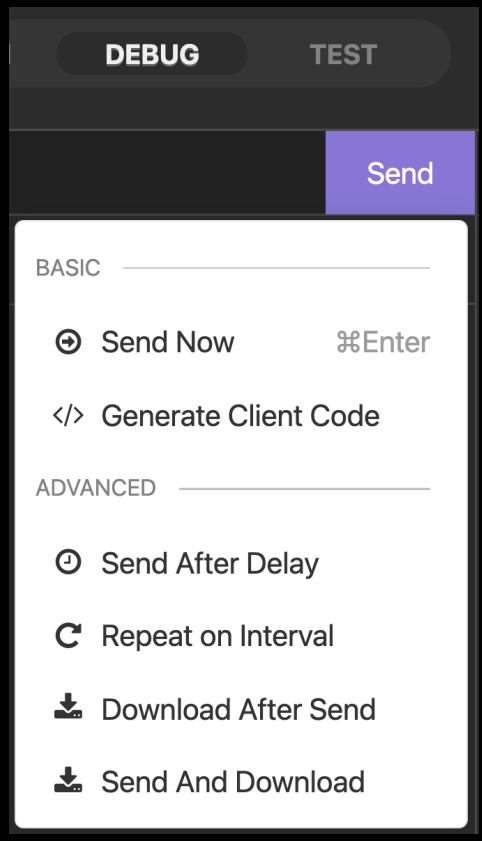


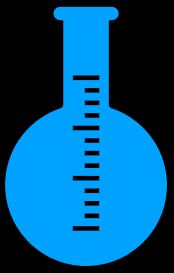
- **Integration Testing**
  - **End-to-End**
  - **Doesn’t have to be last**
  - **May take more time than expected**
  - **Maybe (automated) UI driven**
    - **SDK**
    - **Client Agnostic**

API Automated Testing

- **Testing with Insomnia**
  Debug > Ctrl+Send menu
  Delay, Repeat, etc.
  Plugins
  https://insomnia.rest/plugins
  CLI
  https://insomnia.rest/product/automated-testing







- **Automated Testing with Insomnia**

inso CLI

Lint your spec

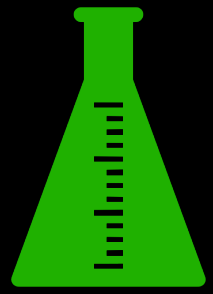
Look for and trap small issues with big effects in syntax and similar

Run Tests

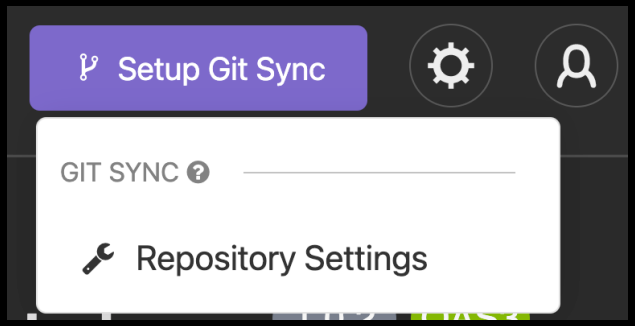
CI/CD on checkin, automated

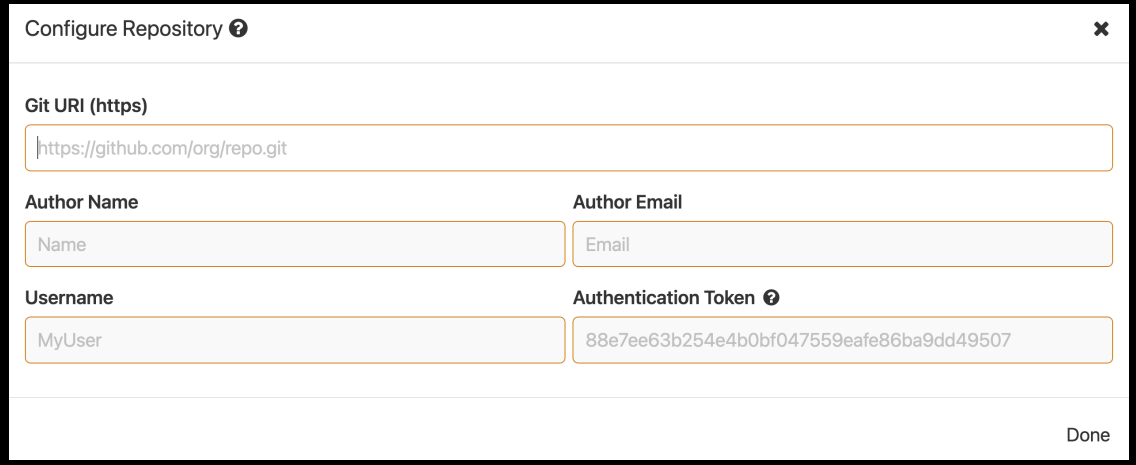
Define config

Generate Code, Documentation

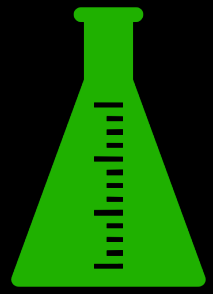


- **CI/CD with Insomnia**
  Create git repo and not URL
  In Insomnia select Setup 'Git Sync' > Repository Settings
  Fill out the form

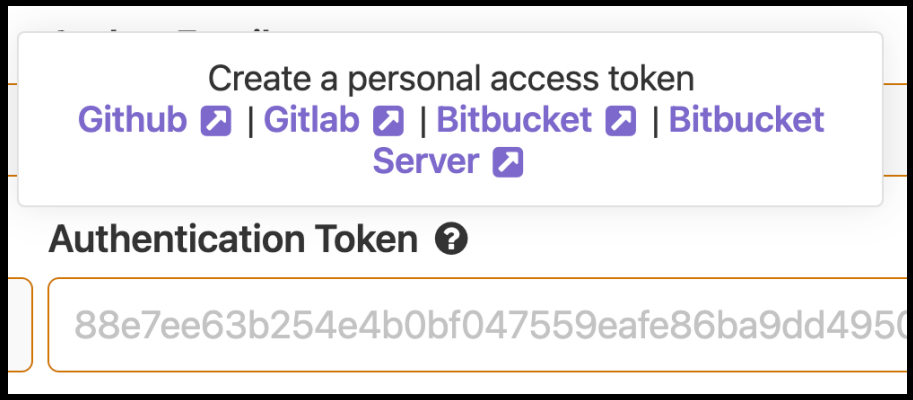


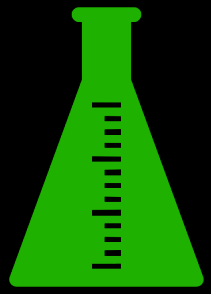


API Automated Testing



- **Token**
  Click ? For links to instructions
  Sync with Github or similar





- **Insomnia CLI**
  Install CLI

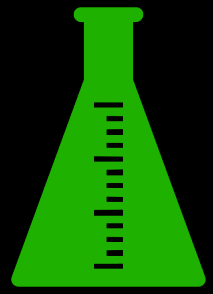
  Download:

  https://github.com/Kong/insomnia/releases/tag/lib@3.14.0

  Docker image:

  https://hub.docker.com/r/kong/inso

API Automated Testing



- **Insomnia CLI**

Execute: inso run test

Select suite

Select environment

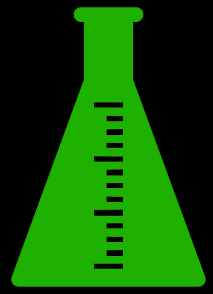
View Results

```
Bear-MBP:~ bearc2020$ inso run test
? Select a document or unit test suite …
Swagger Petstore 1.0.2 spc_6d22e0
  | New Suite 1 uts_07beec
My API spc_e4343f
  | Test 1 uts_276215
```

```
Bear-MBP:~ bearc2020$ inso run test
✔ Select a document or unit test suite · Test 1 - uts_276215
? Select an environment …
OpenAPI env env_env_65edc1
```

```
Bear-MBP:~ bearc2020$ inso run test
✔ Select a document or unit test suite · Test 1 - uts_276215
✔ Select an environment · OpenAPI env - env_env_65edc1


  Test 1
    ✔ Returns 200 (1278ms)


  1 passing (1s)
```

API Automated Testing

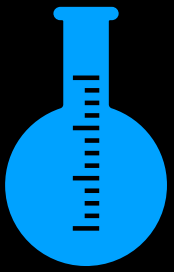


- **Insomnia CLI**
  Note the suite and env identifiers/names
  Run tests directly

```
Bear-MBP:~ bearc2020$ inso run test
✔ Select a document or unit test suite · Test 1 - uts_276215
✔ Select an environment · OpenAPI env - env_env_65edc1

  Test 1
    ✓ Returns 200 (1278ms)

  1 passing (1s)
```

```
Bear-MBP:~ bearc2020$ inso run test uts_276215 --env env_env_65edc1

  Test 1
    ✓ Returns 200 (1144ms)

  1 passing (1s)
```

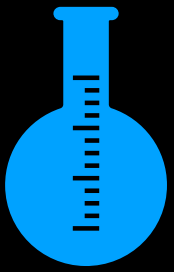
- **Insomnia CLI**
  inso lint spec



```
Bear-MBP:~ bearc2020$ inso lint spec
? Select an API Specification …
Swagger Petstore 1.0.2 spc_6d22e0
My API spc_e4343f
```

```
Bear-MBP:~ bearc2020$ inso lint spec
✔ Select an API Specification · My API – spc_e4343f
No linting errors. Yay!
```

API Automated Testing



- **Insomnia CLI**

inso scripts

Config files

Specify a name and what to run

```
scripts:
  mylint: inso lint spec "My API"
```

Run with script (or w/o if name is unique)

```
Bear-MBP:insomnia bearc2020$ inso mylint
No linting errors. Yay!
Bear-MBP:insomnia bearc2020$ inso script mylint
No linting errors. Yay!
```

Thanks

If you have additional questions,
please reach out to me at:
ashish@datacouch.io