RESTful API
Design and Development

https://www.accelebrate.com
(877) 849-1850 ❖ info@accelebrate.com

Customized Technical Training

Accelebrate
ACCELERATED LEARNING, CELEBRATED RESULTS®

On-Site, Customized Private Training

Don't settle for a one-size-fits-all class! Let Accelebrate tailor a private class to your group's goals and experience. Classes can be delivered at your site or online (or a combination of both) worldwide. Visit our web site at **https://www.accelebrate.com** and contact us at **sales@accelebrate.com** for details.

Public Online Training

Need to train just 1-3 people? Attend one of our regularly scheduled, live, instructor-led public online classes. Class sizes are small (typically 3-6 attendees) and you receive just as much hands-on time and individual attention from your instructor as our private classes. For course dates, times, outlines, pricing, and registration, visit **https://www.accelebrate.com/public-training-schedule.**

Newsletter

Want to find out about our latest class offerings? Subscribe to our newsletter **https://www.accelebrate.com/newsletter**.

Blog

Get insights and tutorials from our instructors and staff! Visit our blog, **https://www.accelebrate.com/blog** and join the discussion threads and get feedback from our instructors!

Learning Resources

Get access to learning guides, tutorials, and past issues of our newsletter at the Accelebrate library, **https://www.accelebrate.com/library**.

Call us for a training quote!

877 849 1850

Accelebrate, Inc. was founded in 2002 with the goal of delivering **private training** that rapidly achieves participants' goals. Each year, our experienced instructors deliver hundreds of classes online and at client sites all over the US, Canada, and abroad. We pride ourselves on our **instructors' real-world experience** and ability to **adapt the training** to your team and their objectives. We offer a wide range of topics, including:

- Angular, React, and Vue
- JavaScript
- Data Science using R, Python, & Julia
- Excel Power Query
- Power BI & Tableau
- .NET & VBA programming
- SharePoint & Microsoft 365
- DevOps & CI/CD
- iOS & Android Development
- PostgreSQL, Oracle, and SQL Server
- Java, Spring, and Groovy
- Agile & IT Leadership
- Web/Application Server Admin
- HTML5 & Mobile Web Development
- AWS, Azure, and Cloud Computing
- Adobe & Articulate
- Docker, Kubernetes, Ansible, & Git
- Software Design and Testing
- AND MORE *(see back)*

"World class training with experts who know their field and have a passion for teaching. Our team is walking away with a wealth of knowledge that we look forward to bringing back to our daily work."

— Keith, Silicon Valley Bank

Like us on Facebook + Follow us on Twitter + Watch us on YouTube

Visit our website for a complete list of courses!

Adobe & Articulate
Adobe Captivate
Adobe Presenter
Articulate Storyline / Studio
Camtasia
RoboHelp

AWS, Azure, & Cloud
AWS
Azure
Cloud Computing
Google Cloud
OpenStack
VMware

Big Data
Alteryx
Apache Spark
Teradata
Snowflake SQL

Data Science and RPA
Blue Prism
Data Literacy
Django
Julia
Machine Learning
MATLAB
Minitab
Python
R Programming
SPSS
Tableau
UiPath

Database & Reporting
BusinessObjects
Crystal Reports
Excel Power Query
MongoDB
MySQL
NoSQL Databases
Oracle
Oracle APEX
Power BI
PivotTable and PowerPivot
PostgreSQL
SQL Server
Vertica Architecture & SQL

DevOps, CI/CD & Agile
Agile
Ansible
Chef
DEI
Docker
Git
Gradle Build System
IT Leadership
ITIL
Jenkins
Jira & Confluence
Kubernetes
Linux
Microservices
OpenShift
Red Hat
Software Design

Java
Apache Maven
Apache Tomcat
Groovy and Grails
Hibernate
Java & Web App Security
JavaFX
JBoss
Oracle WebLogic
Scala
Selenium & Cucumber
Spring Boot
Spring Framework

JS, HTML5, & Mobile
Angular
Apache Cordova
CSS
D3.js
Flutter
HTML5
iOS/Swift Development
JavaScript
MEAN Stack
Mobile Web Development
Node.js & Express
React & Redux
Svelte
Swift
Xamarin
Vue

Microsoft & .NET
.NET Core
ASP.NET
Azure DevOps
C#
Design Patterns
Entity Framework Core
IIS
Microsoft Dynamics CRM
Microsoft Exchange Server
Microsoft 365
Microsoft Power Platform
Microsoft Project
Microsoft SQL Server
Microsoft System Center
Microsoft Windows Server
PowerPivot
PowerShell
VBA
Visual C++/CLI
Visual Studio
Web API

Security
.NET Web App Security
C and C++ Secure Coding
C# & Web App Security
Linux Security Admin
Python Security
Secure Coding for Web Dev
Spring Security

SharePoint
Power Automate & Flow
SharePoint Administrator
SharePoint Developer
SharePoint End User
SharePoint Online
SharePoint Site Owner

SQL Server
Azure SQL Data Warehouse
Business Intelligence
Performance Tuning
SQL Server Administration
SQL Server Development
SSAS, SSIS, SSRS
Transact-SQL

Teleconferencing Tools
Adobe Connect
GoToMeeting
Microsoft Teams
WebEx
Zoom

Web/Application Server
Apache httpd
Apache Tomcat
IIS
JBoss
Nginx
Oracle WebLogic

Other
C++
Go Programming
Project Management
Regular Expressions
Ruby on Rails
Rust
Salesforce
XML

Visit ***www.accelebrate.com/newsletter*** *to sign up and receive our* **newsletters** *with information about* **new courses**, **free webinars**, **tutorials**, *and* **blog articles**.

Call us for a training quote! 877 849 1850 (US/Canada) or +1 678 648 3133

Understanding APIs in Software Delivery: Introduction

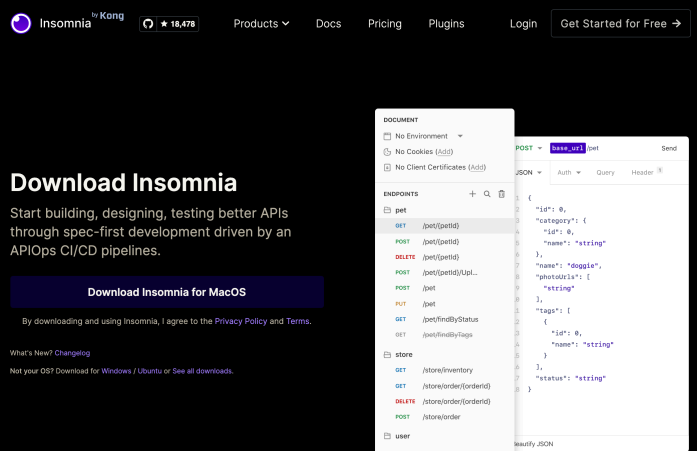
Bear Cahill
Brainwashlnc.com

Freelance Mobile Developer

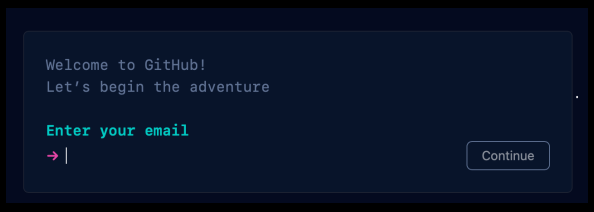
https://www.linkedin.com/in/bearc0025/

Understanding APIs in Software Delivery: Introduction

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



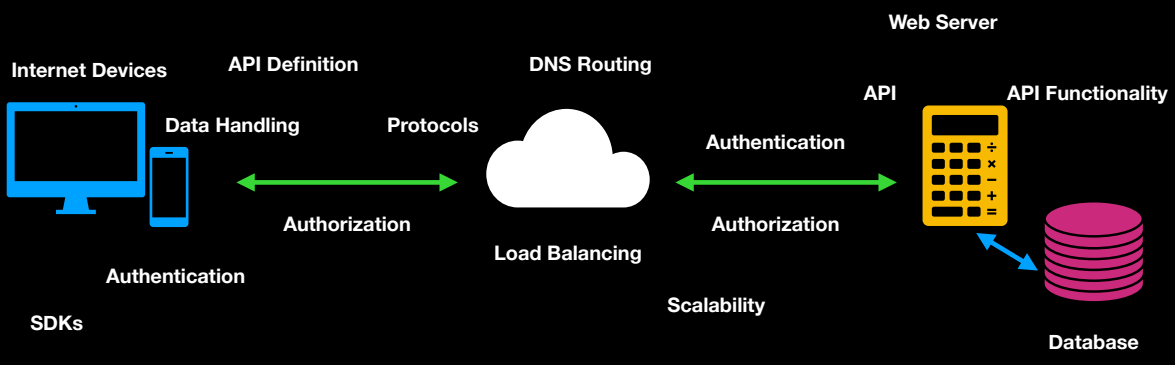
- **Optional: Create Github Account**
  - **https://github.com/signup**



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

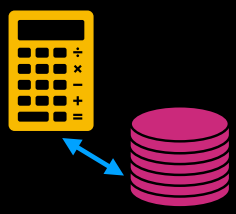
API Design: Introduction

API Design: Introduction



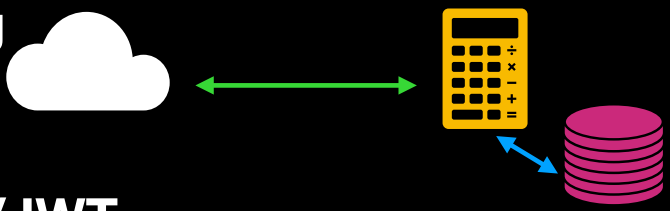
API Design: Introduction

- 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

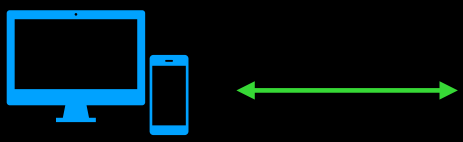
- API Routing & Security
  - DNS
  - Load Balancing
    - Scalability
  - Authentication
    - OAuth, Token/JWT
  - Authorization
    - Permissions



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

API

Web Server

Internet Devices

API Definition

DNS Routing

API

API Functionality

Data Handling

Protocols

Authentication

Authorization

Authorization

Authentication

Load Balancing

SDKs

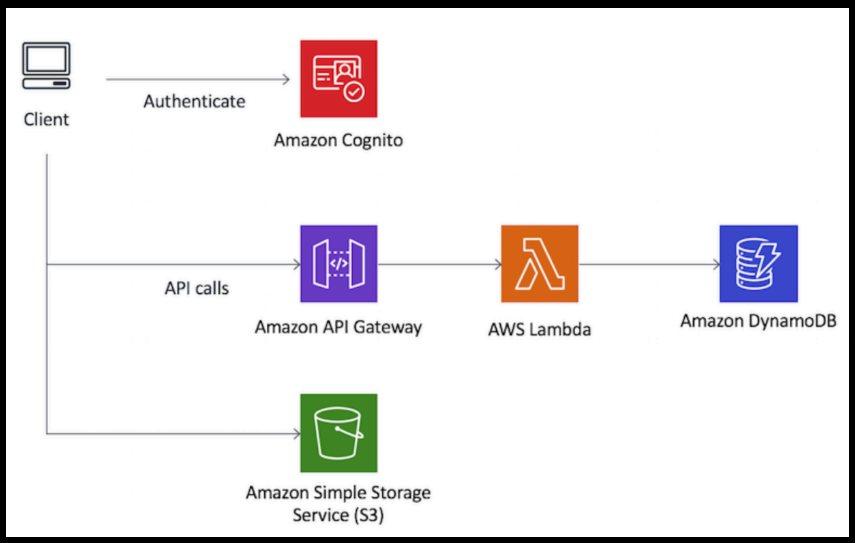
Scalability

Database

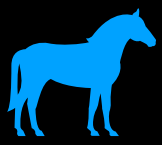
OpenAPI Spec

API Design: Introduction

- **AWS**

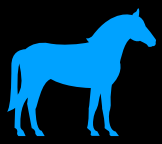


API Design: Introduction

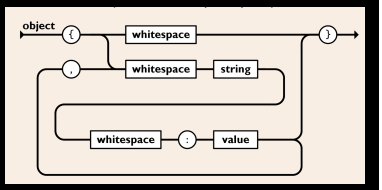


- **Past:**
  - **API Wild West**
    - **URL/Query String, Form Data, etc.**
  - **SOAP/WSDL**
  - **XML, text, etc.**
- **Everything was proprietary, complex**
- **Needed a Standard**

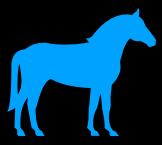
API Design: Introduction



- 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**
  - **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**

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.

- **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)**
  - **https://swagger.io/specification/**

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

API Design: Documenting

- **Open API**
- **More doc/spec:**
- **https://github.com/OAI/OpenAPI-Specification/tree/main/versions**
- **Various vers**

OpenAPI Object

This is the root document object of tl

Fixed Fields

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

Components Object

Holds a set of reusable objects for different aspects of the OAS. All objects defir API unless they are explicitly referenced from properties outside the component

Fixed Fields

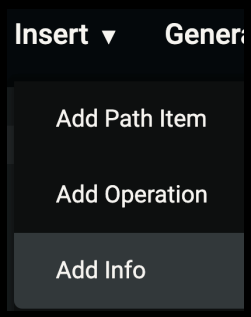
Field Name	Type
schemas	Map[`string` , Schema Object | Reference Object]
responses	Map[`string` , Response Object | Reference Object]
parameters	Map[`string` , Parameter Object | Reference Object]
examples	Map[`string` , Example Object | Reference Object]
requestBodies	Map[`string` , Request Body Object | Reference Object]
headers	Map[`string` , Header Object | Reference Object]

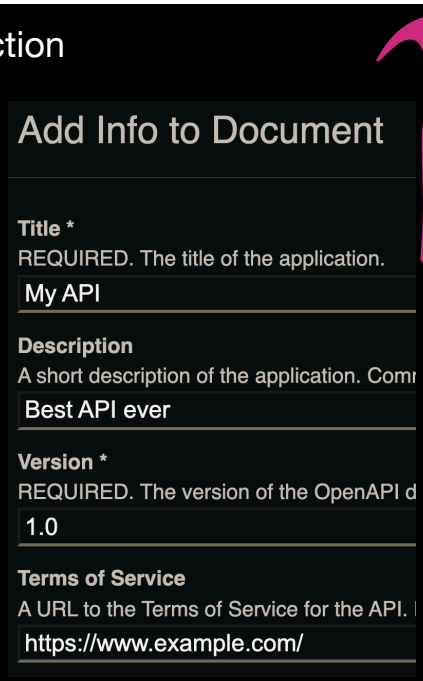
API Design: Introduction

- Editor: Info
- editor.swagger.io
- Start with version
- Errors
- Insert > Info

API Design: Introduction

- Editor
- Insert > Info
  - Enter text in fields
  - License/Contact optional
- Click 'Add Info'





API Design: Introduction

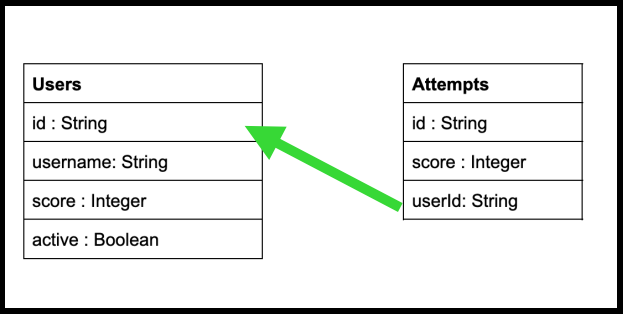
- Text inserted

```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

- Consider the data represented
- Fetch users
  - …/users
- Fetch 1 user
  - …/users/<id>
- Attempts too.
- User's attempts
  - …/users/<id>/attempts
- What is returned?



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

- **Data represented in JSON**
- **…/users/1234abc**

```
{
  "id": "1234abc",
  "username": "bear",
  "score": 55,
  "active": true
}
```

Users
id : String
username: String
score : Integer
active : Boolean

API Design: JSON

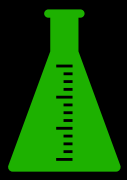
- JSON
  - Object name (User) not included
  - Arrays [ ]
  - Dictionary { }
  - Keys are Strings
  - Values vary
    - Strings, Ints, Bool, Dictionaries, Array

```
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)

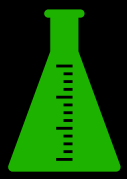
API Design: JSON



- Lab
  - https://jsonlint.com/
  - Create an array of Elements
  - Each element has 4 members:
    - id of type String
    - username of type String
    - score of type Integer
    - active of type Boolean
  - See next slide for hints

Users
id : String
username: String
score : Integer
active : Boolean

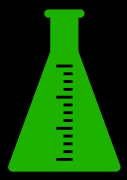
API Design: JSON



- Lab
  - json.org has details
  - Arrays are within [and]
  - Elements are within { and }
  - Element members are key-value pairs
    - “id" : "test"
  - Members and Array elements are comma separated
  - See next slide for example

API Design: JSON

- Lab



```
[{
    "id": "1234abc",
    "username": "bear",
    "score": 55,
    "active": true
}, {
    "id": "xyz123",
    "username": "schmeb",
    "score": 550,
    "active": false
}, {
    "id": "qwerty",
    "username": "jed",
    "score": 553,
    "active": true
}]
```

- 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

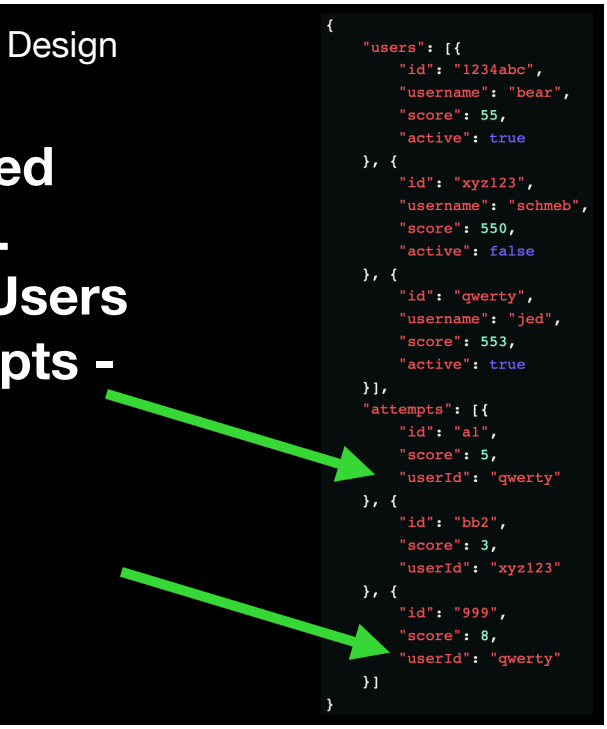
- Key Aspects
  - Nouns - resource based
    - e.g., User, Comment, Post
  - Verbs to URIs
    - e.g., GET, POST
  - Stateless - each request has it all
    - Scaleable - doesn't need same server for next request

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
  - Nouns - resource based
    - Relationships in URL
    - Attempts relates to Users
    - /users/qwerty/attempts - returns 2 comments



```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```


API Design: Design

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


```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```

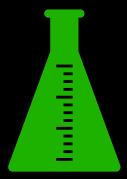
API Design: Design

- Verbs - CRUD
  - GET - read
    - …/users - all
    - …/users/1 - post with id = 1
    - …/attempts - all
    - …/attempts/1 - one
    - …/users/1/attempts

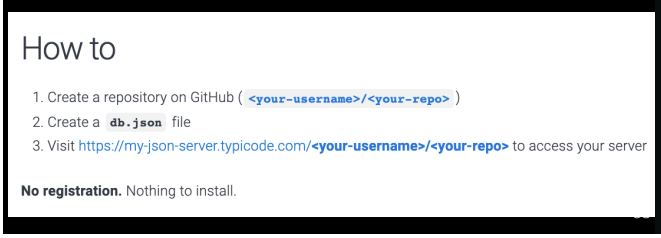

```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```

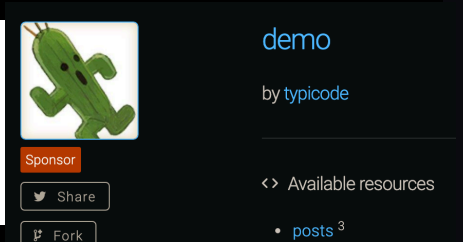

API Design: Design

- Lab: GET
- 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

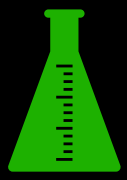



```
{
  "posts": [
    {
      "id": 1,
      "title": "Po
    },
    {
      "id": 2,
      "title": "Po
    },
    {
      "id": 3,
      "title": "Po
    }
  ],
  "comments": [
    {
      "id": 1,
      "body": "som
      "postId": 1
    },
    {
      "id": 2,
      "body": "som
```



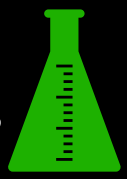
API Design: Design



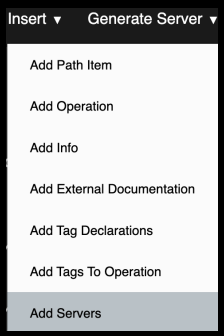
- **Lab: GET - Different db.json**
- **https://my-json-server.typicode.com/bearc0025/api**
- **View db.json file**
- **Via browser:**
  - **GET users, 1 user, attempts, 1 attempt, etc.**

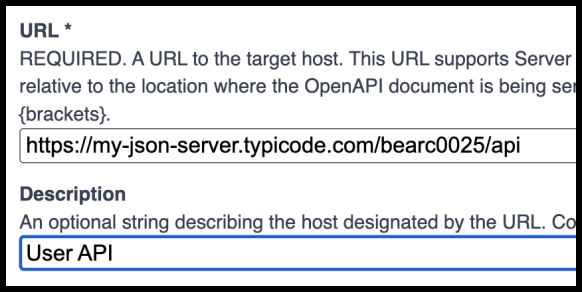

```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
```


API Design: Design



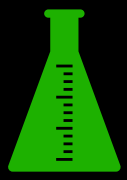
- **Lab: OpenAPI Server**
- **editor.swagger.io > Insert > Add Servers**
- **Fill out form**
- **Click Add Servers (bottom right)**





API Design: Design

- **Lab: OpenAPI Server**
- **Results**

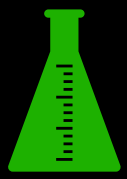


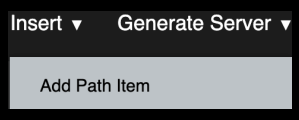
```yaml
servers:
  - url: https://my-json-server.typicode.com/bearc0025/api
    variables: {}
    description: User API
```

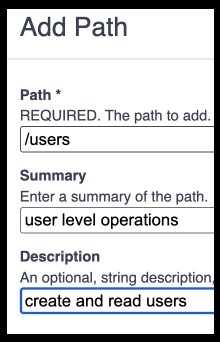
- **End of lab**

API Design: Design

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





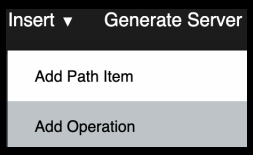


Creates:

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

API Design: Design

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

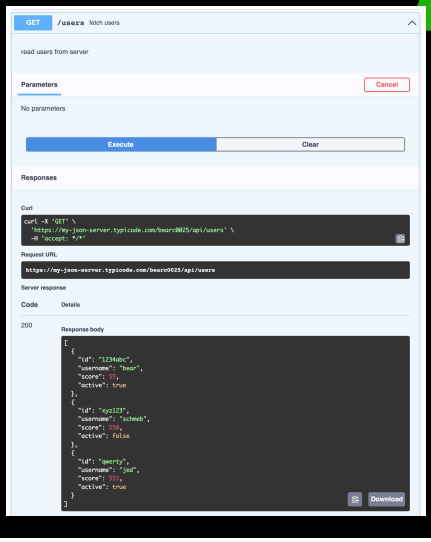


Creates:

```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

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



- **End of lab**

API Design: Design

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


```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```


API Design: Design

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

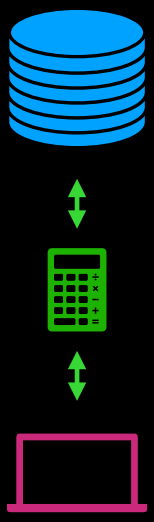

```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```

API Design: Design

- Query String
  - Additional parameters:
    - Filtering
    - Searching
    - Paging
    - Embedding
    - etc.
- …/api/users/1?_embed=attempts


```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```


API Design: Design Concepts

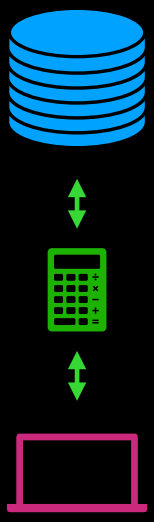


API Design: Design Concepts

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

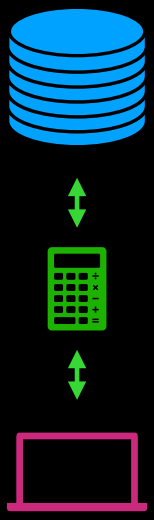
API Design: Design Concepts

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



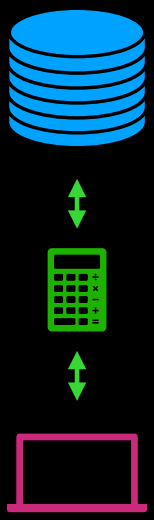
API Design: Design Concepts

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



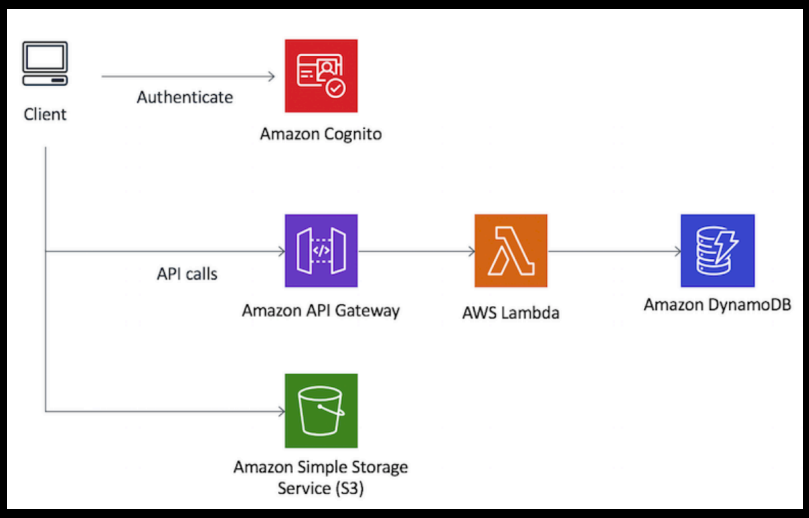
API Design: Design Concepts

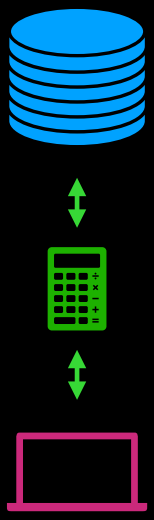
- Separation of Concerns
  - Delivering Data
    - How it is sent: protocol, etc.
    - Important to client
    - Directed by server
      - Usually the same for all clients



API Design: Design Concepts

- **AWS - Separation of Concerns**





API Design: Design

- **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
```

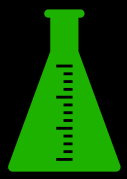
API Design: Design

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

Users
id : String
username: String
score : Integer
active : Boolean

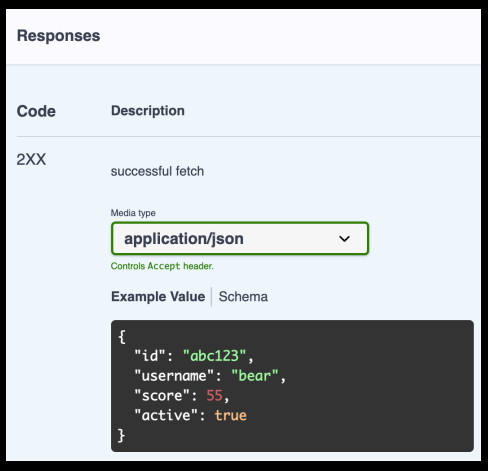
```yaml
get:
  summary: fetch users
  description: read users from server
  operationId: fetchUsers
  responses:
    '2XX':
      description: successful fetch
      content:
        application/json:
          schema:
            type: object
            properties:
              id:
              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


API Design: Design



- 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

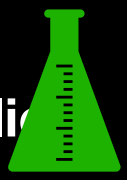
API Design: Design Concepts

- 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

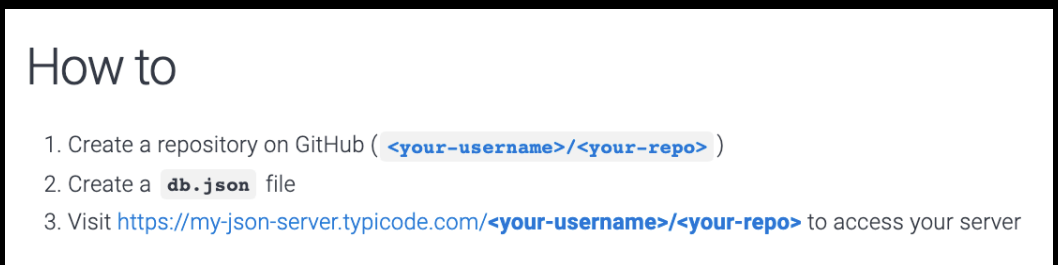
API Design: Design Concepts

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

API Design: Design

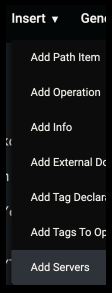


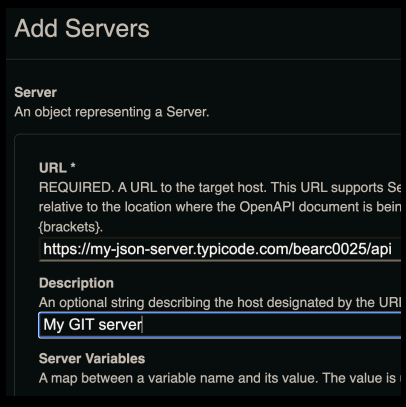
- **Lab**
- **If you have a git account. Create a public repo…**
- **https://my-json-server.typicode.com/**
- **Demo data @ https://github.com/bearc0025/api/blob/main/db.json**



API Design: Design Concepts

- **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**

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: Components

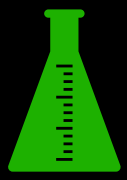
- Many sections for reuse
- Note: schemas
  - Can define schemas
  - Reference in Operations
- Also, respones
  - Reference schemas
  - Reference in Operations

Fixed Fields

Field Name	Type	Description
schemas	Map[`string`, Schema Object | Reference Object]	An object to hold reusable Schema Objects.
responses	Map[`string`, Response Object | Reference Object]	An object to hold reusable Response Objects.
parameters	Map[`string`, Parameter Object | Reference Object]	An object to hold reusable Parameter Objects.
examples	Map[`string`, Example Object | Reference Object]	An object to hold reusable Example Objects.
requestBodies	Map[`string`, Request Body Object | Reference Object]	An object to hold reusable Request Body Objects.
headers	Map[`string`, Header Object | Reference Object]	An object to hold reusable Header Objects.
securitySchemes	Map[`string`, Security Scheme Object | Reference Object]	An object to hold reusable Security Scheme Objects.
links	Map[`string`, Link Object | Reference Object]	An object to hold reusable Link Objects.
callbacks	Map[`string`, Callback Object | Reference Object]	An object to hold reusable Callback Objects.

- https://swagger.io/specification/

API Design: Components



- Create a components section (bottom)
- Add a schemas section
- Add a schema in it named ‘FullUser’
- Copy under “schema” in the ‘2XX’ response
  - Fix indenting
- Replace ‘2XX’ schema with ref

```yaml
responses:
  '2XX':
    description: successful fetch
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/FullUser'
```

```yaml
components:
  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
```

- End of lab

API Design: Design

- Lab
- https://pokeapi.co/
- Fetch data and drill down into more
- Try with limit and offset

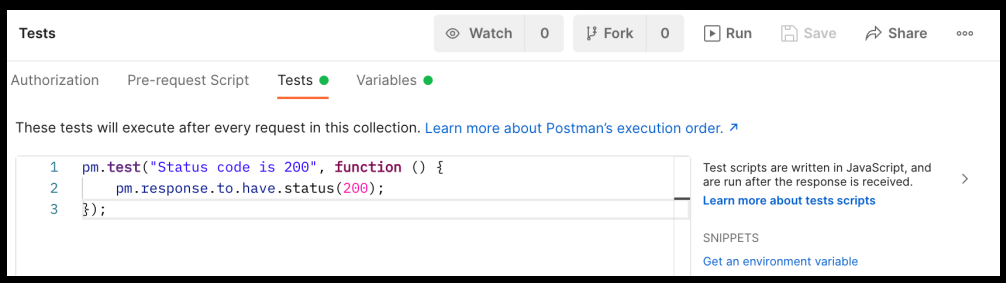
API Design: Design

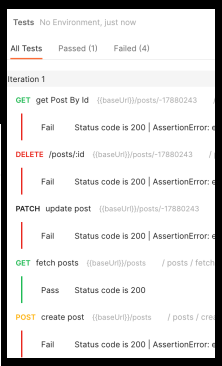
- Lab
- https://swapi.dev/
- Fetch data and drill down into more
- Try with page query string parameter

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

API Design: Testing and Development

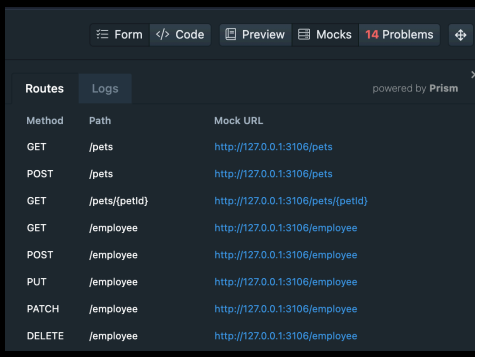
- Test
  - Tools: ReadyAPI, PostMan, Insomnia, StopLight, Paw, etc.
  - Mock endpoints





API Design: Testing and Development

- **Accuracy/Functionality**
- **Performance - scalability**
- **Errors - fail well**
- **Automate when possible**
  - **Including CI/CD**



API Design: Testing and Development

- **Previous steps: Doc, Security, Versioning, etc.**
- **Audience: public, private**
- **Once published, no changes (new vers)**
  - **Maybe not even bug fixes if it changes how the API works**
- **Monitor**
  - **Logs, errors, user input**

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
    - /attempts/yesterday
  - 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

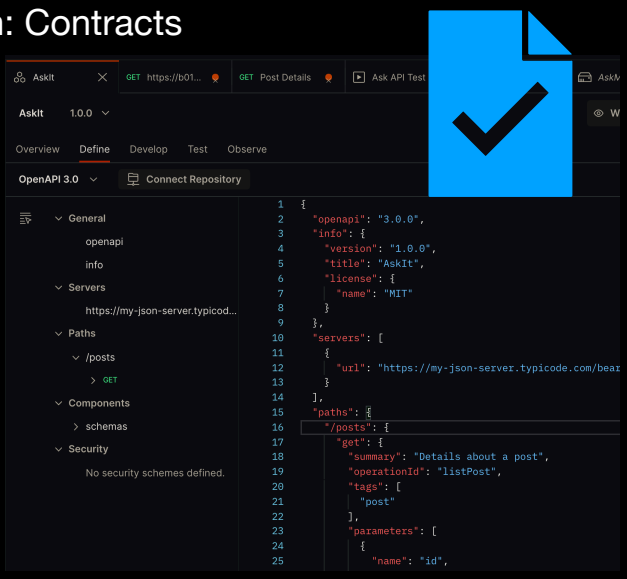
API Design: Contracts

- OpenAPI Documentation
- https://swagger.io/specification/
- Various sections and types

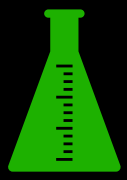
Field Name	Type	Description
openapi	`string`	**REQUIRED**. This string MUST be the semantic version number of the OpenAPI Specification version that the OpenAPI document uses. The `openapi` field SHOULD be used by tooling specifications and clients to interpret the OpenAPI document. This is *not* related 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

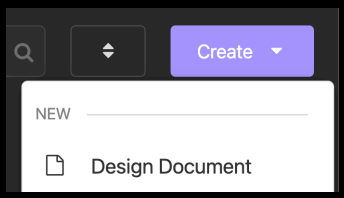
- Tools
  - SwaggerUI/Hub
  - Insomnia
  - OpenAPI Generator
  - Postman
  - PAW
  - Stoplight
  - Swashbuckle
- Can help generate or consume

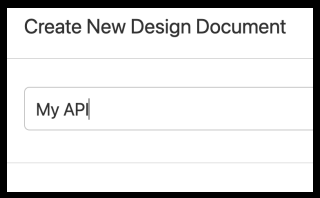


API Design: Insomnia

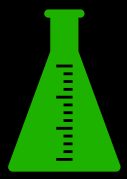


- **Create API in Insomnia**
  - **Open Insomnia**
  - **Click Create > Design Document**
    - **Give it a name and click Create**

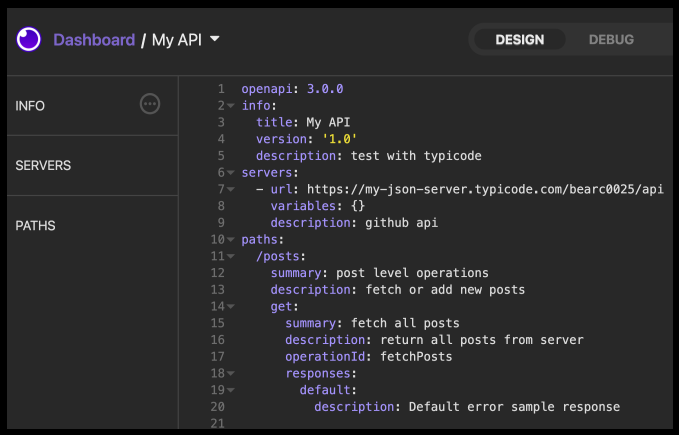




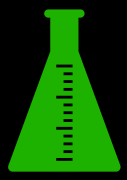
API Design: Insomnia



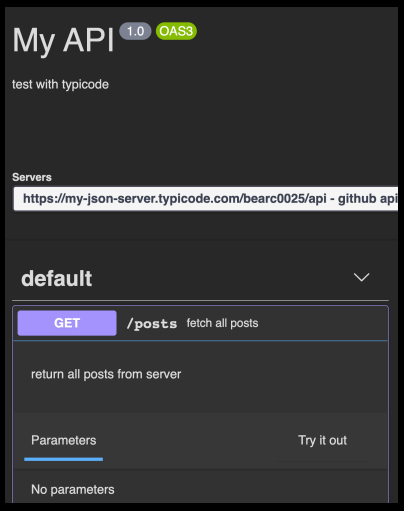
- **Paste your texts from the Swagger editor into the Insomnia editor**



API Design: Insomnia

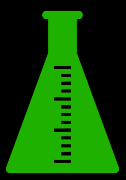


- **Test on the right**



- **End of lab**

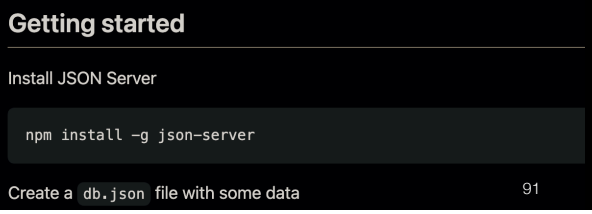
API Design: RESTful API Basics



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

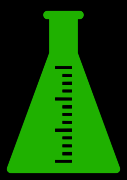
API Design: RESTful API Basics

- **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/bearc0025/api/db**



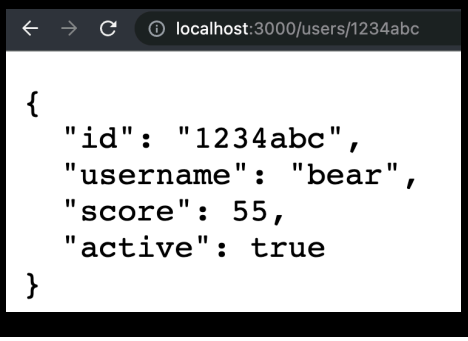

```
{
    "users": [{
        "id": "1234abc",
        "username": "bear",
        "score": 55,
        "active": true
    }, {
        "id": "xyz123",
        "username": "schmeb",
        "score": 550,
        "active": false
    }, {
        "id": "qwerty",
        "username": "jed",
        "score": 553,
        "active": true
    }],
    "attempts": [{
        "id": "a1",
        "score": 5,
        "userId": "qwerty"
    }, {
        "id": "bb2",
        "score": 3,
        "userId": "xyz123"
    }, {
        "id": "999",
        "score": 8,
        "userId": "qwerty"
    }]
}
```


API Design: RESTful API Basics

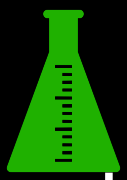


- **Lab**
- **json-server --watch db.json**
- **http://localhost:3000/users**
- **http://localhost:3000/users/1**

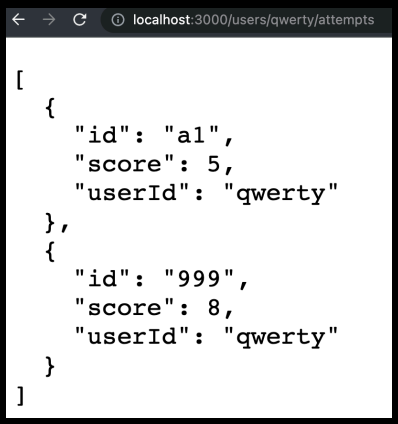
```
Bear-MBP:json-server bearc2022$ json-server --watch db.json

  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3000/users
  http://localhost:3000/attempts

  Home
  http://localhost:3000
```



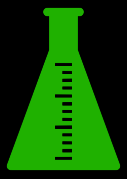
API Design: RESTful API Basics

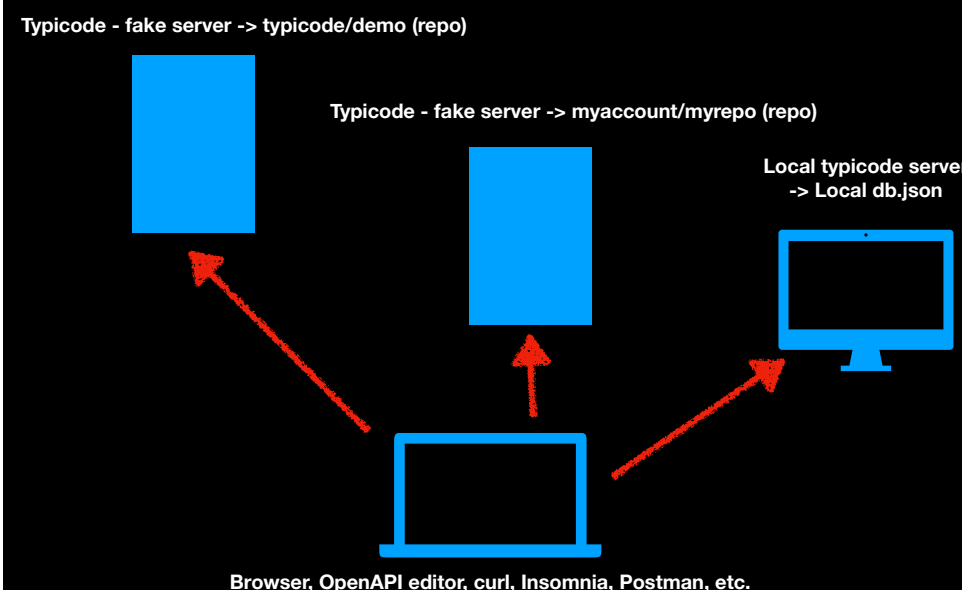


- **Lab**
- **http://localhost:3000/users/qwerty/attempts**

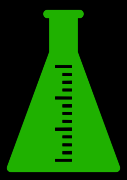


API Design: RESTful API Basics

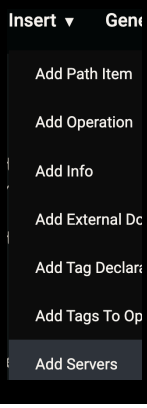




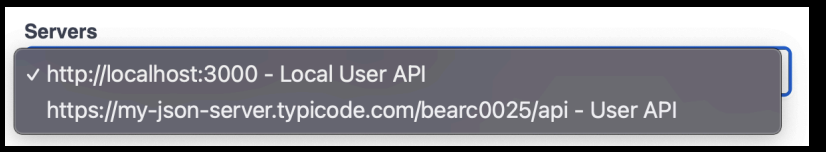
API Design: RESTful API Basics



- **Lab**
- **Add Server to Swagger editor**
- **Via menu or edit text**

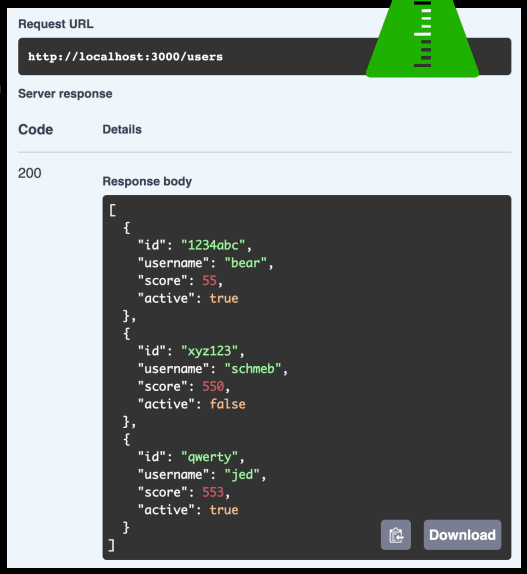



```
servers:
  - url: http://localhost:3000
    variables: {}
    description: Local User API
  - url: https://my-json-server.typicode.com/bearc0025/api
    variables: {}
    description: User API
```



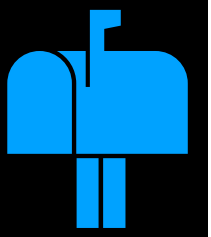
API Design: RESTful API Basics

- **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

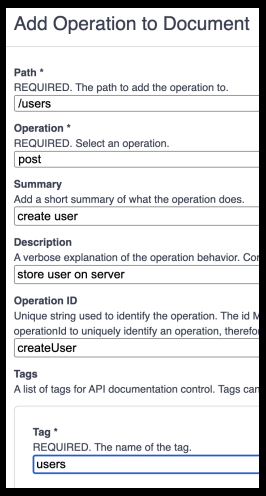
Describes a single request body.

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
```

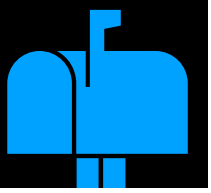

API Design: RESTful API Basics

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

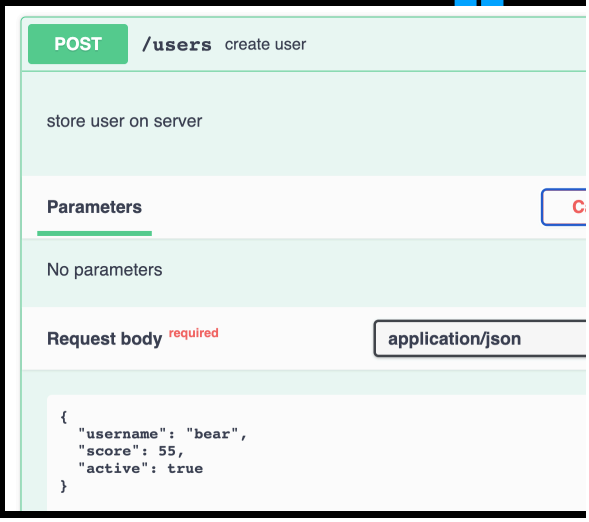
```yaml
type: object
properties:
  username:
    type: string
    example: bear
  score:
    type: integer
    example: 55
  active:
    type: boolean
    example: true
required:
  - username
```

```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
- Test in documentation
- Verify response



- End of lab

API Design: RESTful API Basics

- Lab: Add Response to POST
- Add PostNewUser to schemas:

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

- Add ref in post method requestBody:

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

- 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'

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

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

  - End of lab

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

- 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'
```

```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'
```

- End of lab

- 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

- Server Variables
  - Allow for various values
    - Ports
    - Schemes
    - Versions

API Design

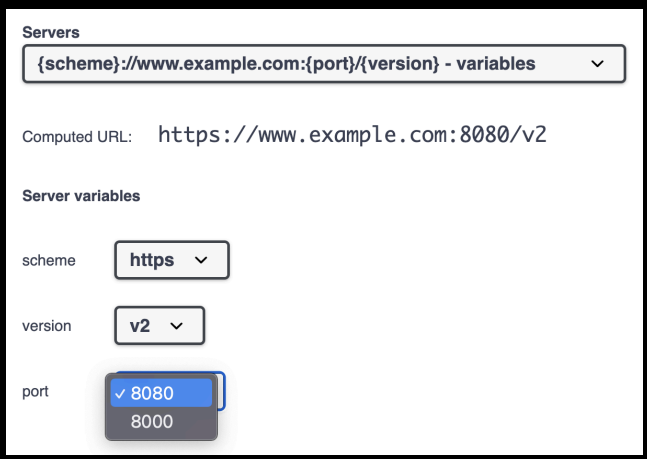
- **Server Variables**
  - **Insert > Add Servers**
  - **Variables within { }**


```
{scheme}://www.example.com:{port}/{version}
```

API Design

- **Server Variables**


```yaml
- url: '{scheme}://www.example.com:{port}/{version}'
  variables:
    scheme:
      default: https
      enum:
        - http
        - https
      description: 'protocol scheme'
    version:
      default: v2
      enum:
        - v2
        - v1
      description: 'version of api'
    port:
      default: '8080'
      enum:
        - '8080'
        - '8000'
      description: 'server port'
```



API Design: Documenting

API Design

- **OpenAPI: Paths and Parameters**
  - **May be templated:**
    - **/users/{userId}**
    - **/users/{userId}/attempts**
  - **OpenAPI specifies parameters**
    - **name**
    - **in: path, query, header, cookie**
    - **schema: typically integer or string**

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

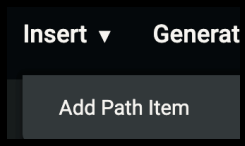
API Design

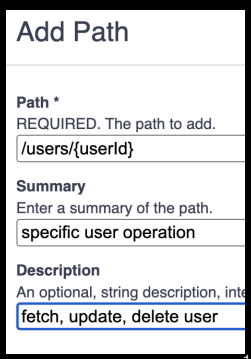
- OpenAPI: Parameters
  - in:
  - path - URL
  - query - Query String
  - header - request header
  - cookie - Cookie

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


API Design

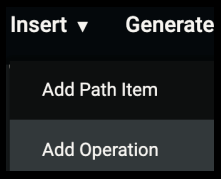
- Lab: /users/{userId}
  - Add /users/{userId}
  - Add GET method
    - Include id parameter of type Integer

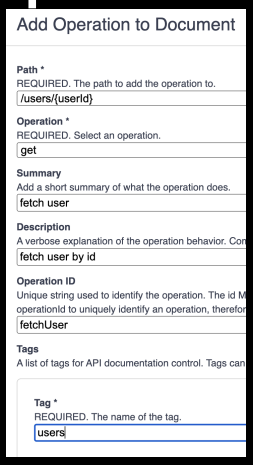





```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
```

- **Lab: /users/{userId}**
  - **Add GET Operation**

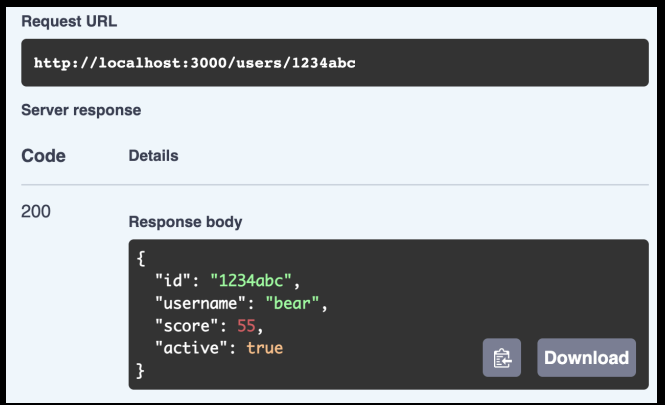





```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
  get:
    summary: fetch user
    description: fetch user by id
    operationId: fetchUser
    responses:
      default:
        description: Default error sample response
    tags:
      - users
```


- **Lab: /users/{userId}**
  - **Add GET parameters section (type in editor)**
  - **Try it out**

```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
  get:
    summary: fetch user
    description: fetch user by id
    operationId: fetchUser
    parameters:
      - name: userId
        in: path
        required: true
        schema:
          type: integer
          minimum: 0
          example: 1
```



API Design

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

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


API Design

- 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
```

API Design

- 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
```


API Design

- 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: /users/{userId}
  - Put parameters section on path level
  - Applies to all operations for this path

```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
  parameters:
    - name: userId
      in: path
      required: true
      schema:
        type: string
        example: 1234abc
```

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

```yaml
components:
  parameters:
    UserId:
      name: userId
      in: path
      required: true
      schema:
        type: string
        example: 1234abc
```

```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
  parameters:
    - $ref: '#/components/parameters/UserId'
```


- Lab: /users/{userId}
  - Put parameter in parameters section of components
  - Refer from operation /user/{userId}

```yaml
components:
  parameters:
    UserId:
      name: userId
      in: path
      required: true
      schema:
        type: string
        example: 1234abc
```

```yaml
/users/{userId}:
  summary: specific user operation
  description: fetch, update, delete user
  parameters:
    - $ref: '#/components/parameters/UserId'
```

  - End of lab

- 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)

- 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
```

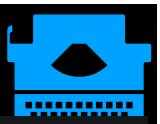
API Design

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

```
/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
```

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


API Design



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

```
/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
```

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

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
```


API Design

- Styles: label - object
  - explode: false

RL

//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
```


API Design

- 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
```

API Design

- Query Parameters
  - Styles
    - form
    - spaceDelimited
    - pipeDelimited
    - deepObject

API Design

- Query Parameter Styles
  - form

```
/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
```

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`

  - Explode true

Request URL

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

API Design

- 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

API Design

- 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
```

API Design

- **Header, Cookie Serialization**
- **See https://swagger.io/docs/specification/serialization/**

Header Parameters

Header parameters always use the `simple` style, that is, comma-separated values. This corresponds to the `{param_name}` URI template. An optional `explode` keyword controls the object serialization. Given the request header named `X-MyHeader`, the header value is serialized as follows:

style	explode	URI template	Primitive value X-MyHeader = 5	Array X-MyHeader = [3, 4, 5]	Object X-MyHeader = {"role": "admin", "firstName": "Alex"}
simple *	false *	{id}	X-MyHeader: 5	X-MyHeader: 3,4,5	X-MyHeader: role,admin,firstName,Alex
simple	true	{id*}	X-MyHeader: 5	X-MyHeader: 3,4,5	X-MyHeader: role=admin,firstName=Alex

* Default serialization method

Cookie Parameters

Cookie parameters always use the `form` style. An optional `explode` keyword controls the array and object serialization. Given the cookie named `id`, the cookie value is serialized as follows:

style	explode	URI template	Primitive value id = 5	Array id = [3, 4, 5]	Object id = {"role": "admin", "firstName": "Alex"}
form *	true *		Cookie: id=5		
form	false	id={id}	Cookie: id=5	Cookie: id=3,4,5	Cookie: id=role,admin,firstName,Alex

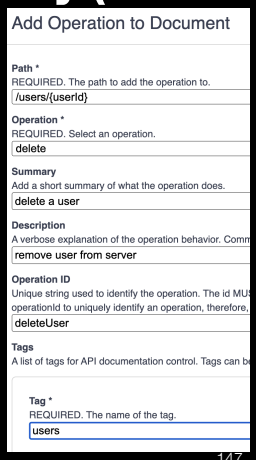
* Default serialization method

API Design

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

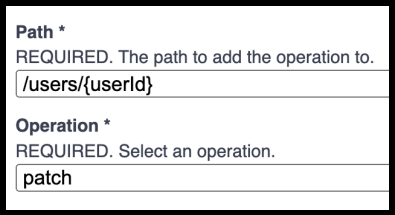
```
/posts/{postId}:
  summary: individual post operations
  description: fetch, delete, update
  parameters:
    - $ref: '#/components/parameters/PostId'
```

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



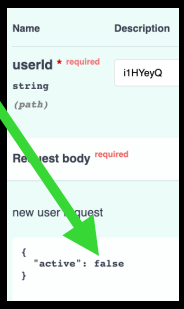

```yaml
delete:
  summary: delete a user
  description: remove user from server
  operationId: deleteUser
  responses:
    default:
      description: Default error sample response
  tags:
    - users
```


- **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**




```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
```

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



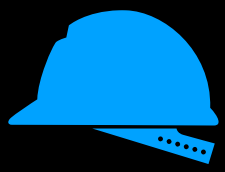
API Design: Design First

- 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

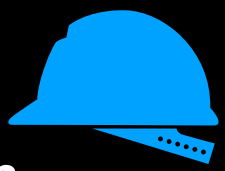
- What could go wrong?
  - Reality - backend/db isn't as expected when implemented
  - Client needs aren't served
  - Large/many queries for little data
- Good for exposing data to unknown clients and uses

API Design: Code First



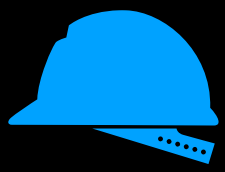
API Design: Code First

- **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

API Design: Automated Testing

- **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/**

.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);
```

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

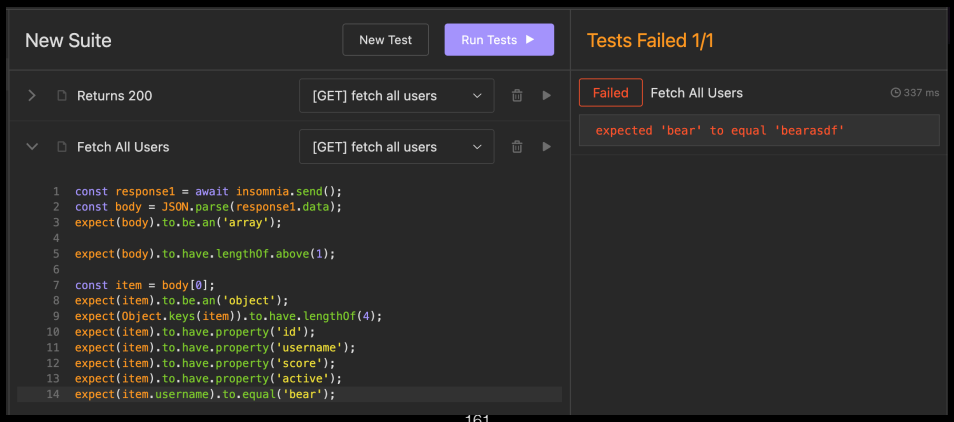
The following are provided as chainable getters to improve the readability of your assertions.

Chains

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

API Design: Automated Testing

- **Insomnia**
  - **Create Test Suite**
  - **Add tests and 'expects'**



API Design: Automated Testing

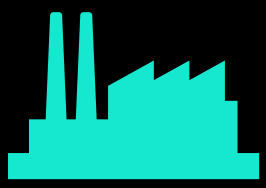
- **Insomnia**
  - **Create Test Suite**
  - **Add tests and 'expects'**

```
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('postId');
```

API Design: Automated Testing

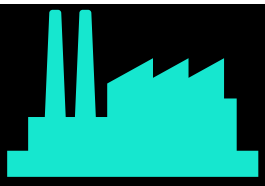
- Insomnia and other tools can…
  - Lint your spec
    - Look for and trap small issues with big effects in syntax and similar
  - Run Tests
    - CI/CD on checkin, automated
  - Define config
  - Generate Code, Documentation

API Design: Legacy Systems



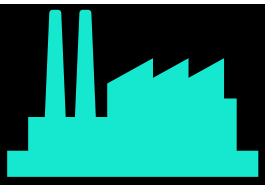
- What is the Legacy system?
- How to map to REST nouns
  - What are the data?
  - Tables? Columns? Join?
  - e.g., Report generation -> /report
- Focus on API, state (not functionality)
- API might be a facade
- Treat the client and server as equals

API Design: Legacy Systems



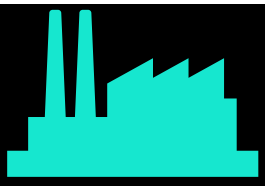
- Is the Legacy system based on database tables?
  - Nouns may be based on tables
  - Verbs convert straight from CRUD
  - Functionality can be minimal
  - Authentication always a concern

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

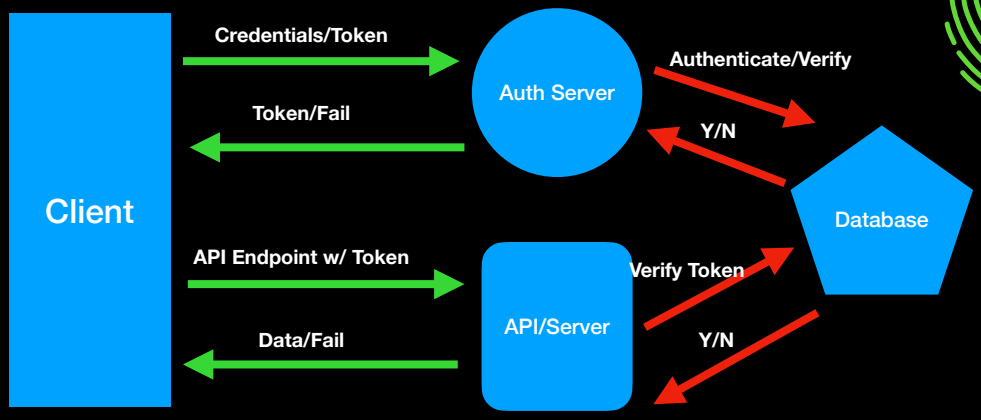
- Authentication vs Authorization
  - Client needs to be authenticated
    - e.g., username + password
  - Results in authorization
    - e.g., token
    - Has certain permissions (and not others)

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

API Design: Security by Design

- Authenticate and Authorize



API Design: Security by Design

- **securityDefinitions is now 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
```

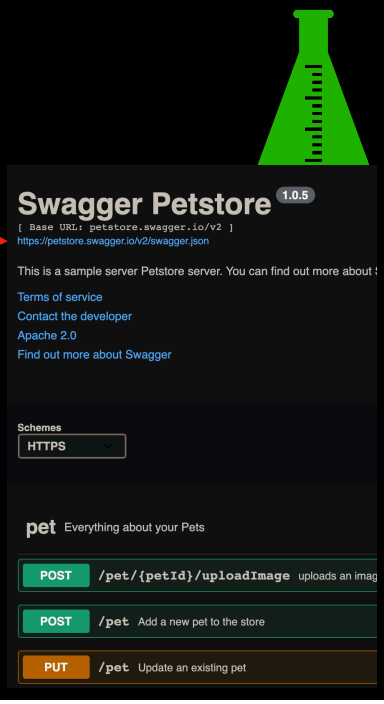

API Design: Security by Design

- **Example: Cognito**

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

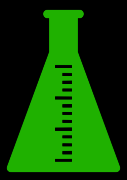
```yaml
paths:
  /v2/alarm:
    get:
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/sc
      security:
      - MyAppAuth: []
```

- **Lab: Pet Store API**
  - https://petstore.swagger.io/
  - Load JSON
  - Copy text to jsonlint.com
    - Validate and review

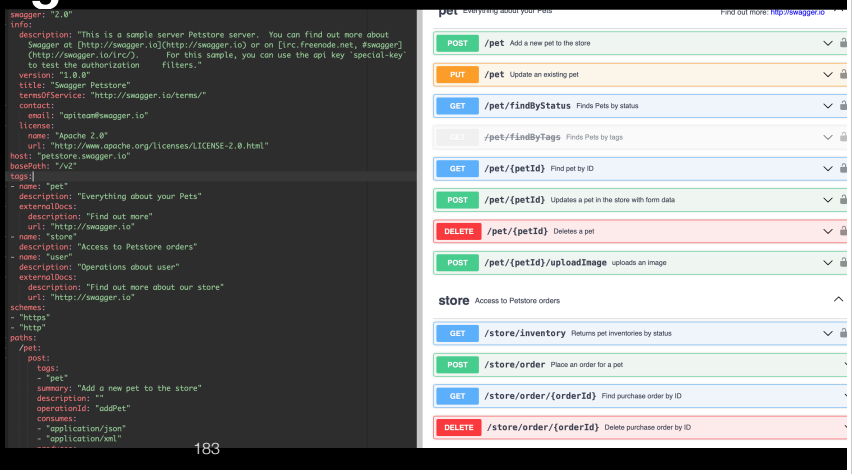


- **Lab: Pet Store API**
  - Notice that it's swagger 2.0 JSON
  - But largely similar in content

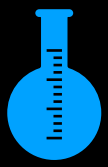
API Design



- **Lab: Pet Store API**
  - **Copy JSON into editor.swagger.io**
  - **Review the generated docs**
  - **Test**
  - **End of lab**

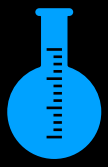


API Automated Testing



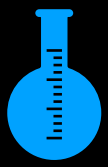
- **Principles**
  - **Types of testing related to API:**
    - **UI**
    - **Functional**
    - **Security**
    - **Performance/Load**
    - **Error**
    - **Validation - end-to-end**

API Automated Testing



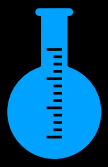
- Principles
  - Types of testing related to API:
    - UI
      - Creating Requests
      - Data format
      - Receiving Responses
    - May be client (not yours)
    - May be SDK

API Automated Testing



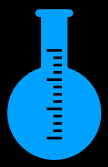
- Principles
  - Types of testing related to API:
    - Functional
      - Back end
      - Code logic
      - Request Processing
      - Data Gathering, Formatting
      - Response Creation

API Automated Testing



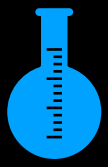
- Principles
  - Types of testing related to API:
    - Security
      - Authentication
      - Authorization
      - Access Prevention

API Automated Testing



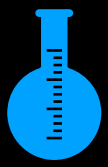
- Principles
  - Types of testing related to API:
    - Performance/Load
      - Process Speed
      - Scalability
      - Cost
      - DOS Security
      - Response Time

API Automated Testing



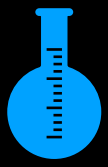
- Principles
  - Types of testing related to API:
    - Error
      - Fail Gracefully
      - Appropriate Status/Error Codes
      - Messages
      - No Secrets
      - Logging

API Automated Testing



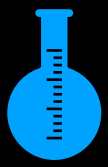
- Best Practices
  - Use Realistic Data
    - Mock Data
    - Real Data
    - All Sizes
    - Include Errors

API Automated Testing



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

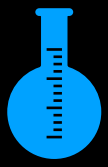
API Automated Testing



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

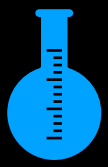
```
(100, 0.5)
(200, 0.75)
(300, 0.85)
(0, 0.0)
```

API Automated Testing



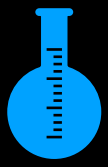
- Best Practices
  - Logging/Analytics
    - Store API traffic
    - Analyze for changes
    - Helps with bug research

API Automated Testing



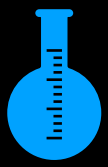
- Best Practices
  - Performance Minded
    - Functional Tests can be reused
      - Performance
      - Scalability
      - Security

API Automated Testing



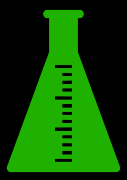
- Best Practices
  - Automation
    - Time $aving
    - Reliable/Repeatable
    - Agnostic to other software/processes
    - Right tool for the right job
      - Insomnia, Postman, Stoplight, etc.
      - CI/CD

API Automated Testing

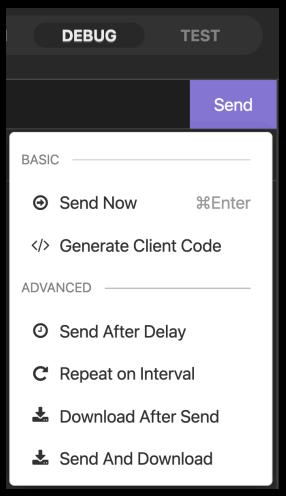


- Integration Testing
  - End-to-End
  - Doesn’t have to be last
  - May take more time than expected
  - Maybe (automated) UI driven
    - SDK
    - Client Agnostic

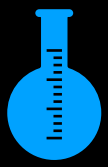
API Automated Testing



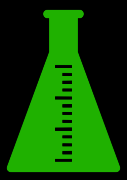
- Testing with Insomnia
  - Debug > Ctrl+Send menu
    - Delay, Repeat, etc.
  - Plugins
    - https://insomnia.rest/plugins
  - CLI
    - https://docs.insomnia.rest/inso-cli/install



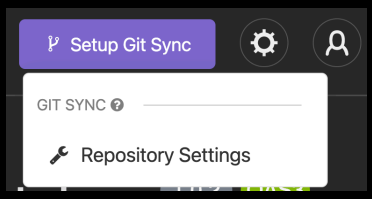
API Design: Automated Testing

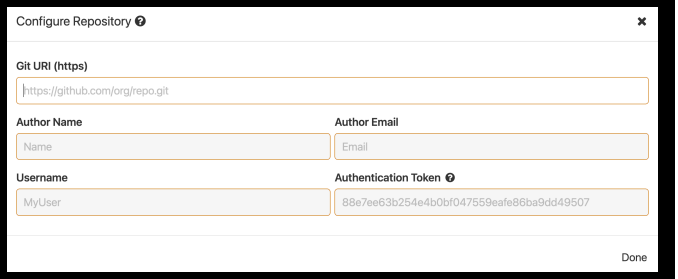


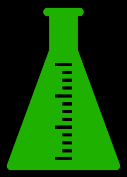
- Automated Testing with Insomnia
- inso CLI
  - Lint your spec
    - Look for and trap small issues with big effects in syntax and similar
  - Run Tests
    - CI/CD on checkin, automated
  - Define config
  - Generate Code, Documentation



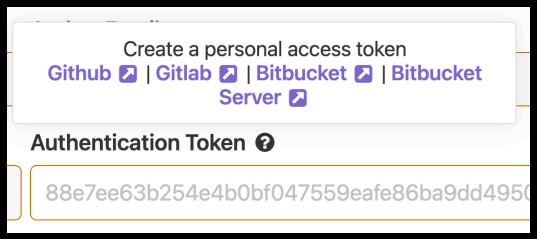
- CI/CD with Insomnia
  - Create git repo and not URL
  - In Insomnia select Setup 'Git Sync' > Repository Settings
  - Fill out the form



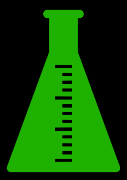




- Token
  - Click ? For links to instructions
  - Sync with Github or similar

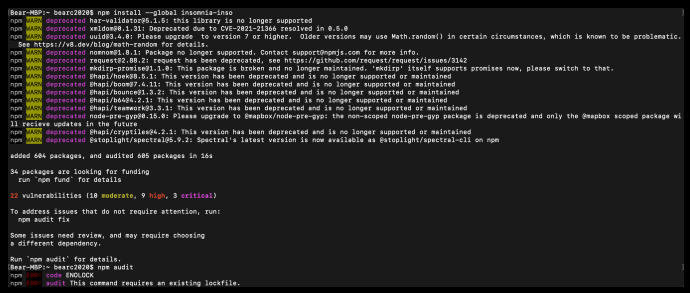


API Automated Testing

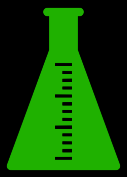


- **Insomnia CLI**
  - **Install CLI:**
  - **https://www.npmjs.com/package/insomnia-inso**

```
npm install --global insomnia-inso
```



API Automated Testing



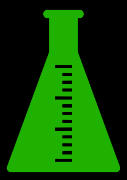
- **Insomnia CLI**
  - **Execute: inso run test**
    - **Select suite**
    - **Select environment**
    - **View Results**

```
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

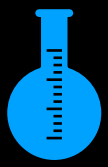


- **Insomnia CLI**
  - **Note the suite and env identifiers/names**
  - **Run tests directly**

```
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)
```


API Automated Testing

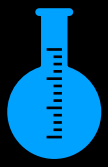


- **Insomnia CLI**
  - **inso lint spec**

```
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



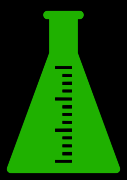
- Insomnia CLI
  - inso scripts
  - Config files
    - Specify a name and what to run

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

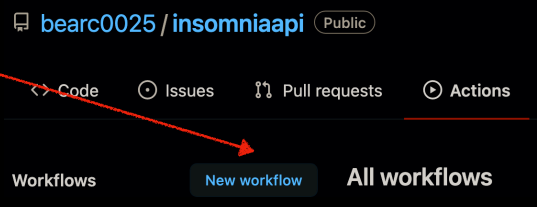
    - Run with script (or w/o if name is unique)

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

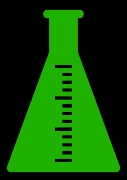

API Automated Testing



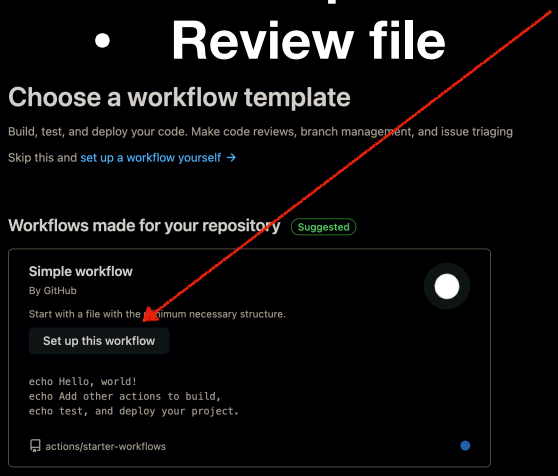
- Insomnia CI/CD Tests
  - e.g., Github Actions
  - https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
  - "New Workflow"



API Automated Testing

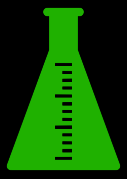


- Choose template
  - "Set up this workflow"
  - Review file




```yaml
# This is a basic workflow to help you get started with Actions

name: CI

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the master branch
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - uses: actions/checkout@v2

      # Runs a single command using the runners shell
      - name: Run a one-line script
        run: echo Hello, world!

      # Runs a set of commands using the runners shell
      - name: Run a multi-line script
        run: |
          echo Add other actions to build,
          echo test, and deploy your project.

```

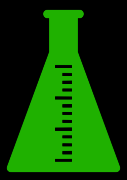

API Automated Testing



- https://docs.insomnia.rest/inso-cli/continuous-integration
- Copy 'steps' from Insomnia example at site above
- Replace command parameters with your project names/ids

```yaml
# .github/workflows/test.yml

name: Test

jobs:
  Linux:
    name: Validate API spec
    runs-on: ubuntu-latest
    steps:
      - name: Checkout branch
        uses: actions/checkout@v1
      - uses: kong/setup-inso@v1
        with:
          inso-version: 2.4.0
      - name: Lint
        run: inso lint spec "Designer Demo" --ci
      - name: Run test suites
        run: inso run test "Designer Demo" --env UnitTest --ci
      - name: Generate declarative config
        run: inso generate config "Designer Demo" --type declarative --ci
```

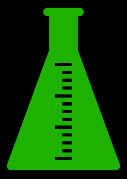
API Automated Testing



- Paste steps into template
  - Optionally remove pull request run in 'on' section
  - On push…
    - Checkout
    - Install
    - Lint
    - Tests
    - Generate config

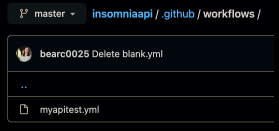
```yaml
# This is a basic workflow to help you get started with Actions

name: CI

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the master branch
  push:
    branches: [ master ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      - name: Checkout branch
        uses: actions/checkout@v1
      - uses: kong/setup-inso@v1
        with:
          inso-version: 2.4.0
      - name: Lint
        run: inso lint spec "My API" --ci
      - name: Run test suites
        run: inso run test uts_276215 --env "OpenAPI env" --ci
      - name: Generate declarative config
        run: inso generate config "My API" --type declarative --ci
```


API Automated Testing

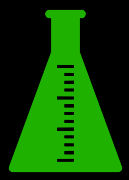


- Change the name of the file

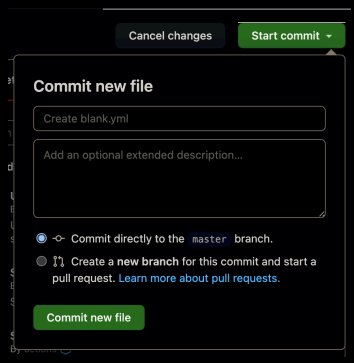
- When saved, .github/workflows with that file will be added to your repo



API Automated Testing

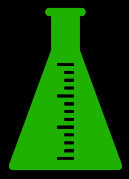


- Commit the file
- Add name and comments as you like
- 'Commit new file'



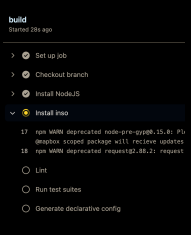
- Pull in Insomnia to keep in sync!

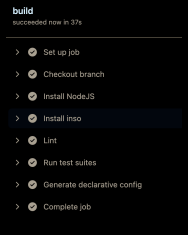
API Automated Testing



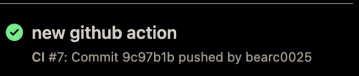
- Go back to 'Actions' and see your action is in progress

- Click it for details

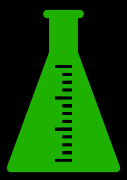




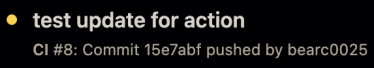
- Actions shows status



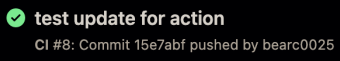
API Automated Testing

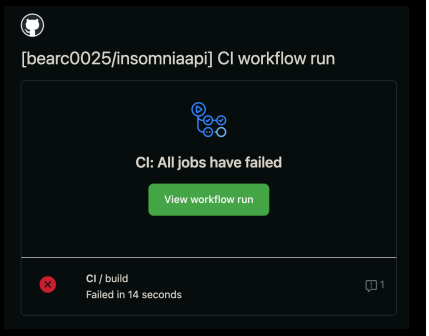


- Make a change in Insomnia
- Commit and Push
- Verify action runs



- Verify success (or failure)





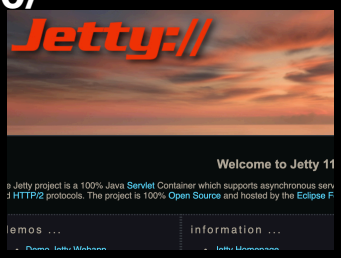
- End of lab

Tutorials

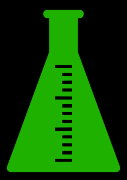
Jetty Web API Servlet

API Design: Java Server and Client

- **Lab: Java Server**
- **Note: This is not meant to be an exhaustive st by-step for configuring Jetty but may be helpful:**
  - **Jetty: https://www.eclipse.org/jetty/**
  - **Unzip**
  - **Set env var:**
    - **JETTY_HOME=/Users/bearc2020/ Downloads/jetty-home-11.0.5**
  - **Start server:**
    - **java -jar start.jar**
  - **Load http://localhost:8080/**

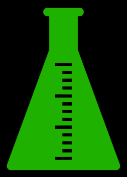


API Design: Java Server and Client

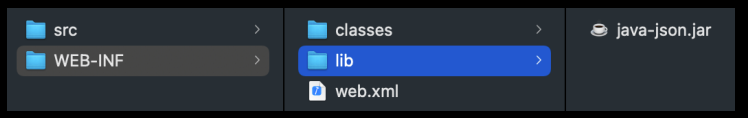


- On same level as jetty-home-<ver>
  - Create jetty-base dir on same level as jetty-home…
  - Run:
  - java -jar start.jar jetty.base=/path/to/your/jetty-base --add-module=server,http,deploy,annotations
  - Change dir to your jetty-base and run:
  - java -jar /path/to/your/jetty-home/start.jar --add-module=server,http,deploy,annotations

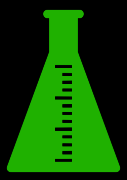
API Design: Java Server and Client



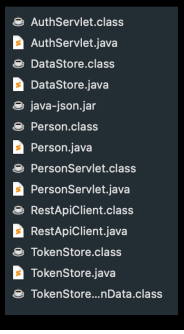
- Under the jetty-base/webapps dir
- Create a PersonServlet dir
- Inside PersonServlet create src and WEB-INF dirs
  - src is for source code but can be located anywhere
  - WEB-INF is for configuration, classes and libraries



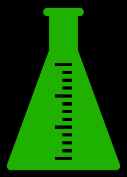
API Design: Java Server and Client



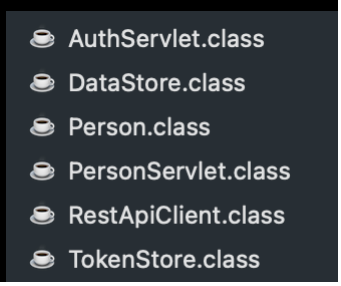
- **Download java-json.jar from here:**
- **java-json/java-json.jar.zip**
- **Copy that to the src and WEB-INF/lib directories**
- **Optional: Copy various source files into the src directory**



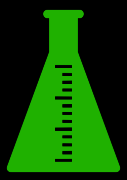
API Design: Java Server and Client



- **Compile the various .java files e.g.,**
- **javac -cp <path to jetty home>/lib/jetty-jakarta-servlet-api-5.0.2.jar:./java-json.jar:. <filename>.java**
  - **NOTE: Not all java files require the serlet api and json jar files.**
- **Copy the class files into**
  - **WEB-INF/classes**



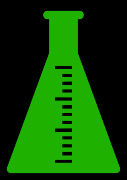
API Design: Java Server and Client



- **Create the web.xml file under the WEB-INF dir**
  - **Boilerplate…**

```
<web-app
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
```

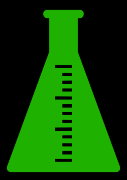

API Design: Java Server and Client



- **Servlet(s)…**

```
<servlet>
    <servlet-name>PersonServlet</servlet-name>
    <servlet-class>PersonServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>PersonServlet</servlet-name>
    <url-pattern>/people/*</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>AuthServlet</servlet-name>
    <servlet-class>AuthServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>AuthServlet</servlet-name>
    <url-pattern>/auth</url-pattern>
</servlet-mapping>
```

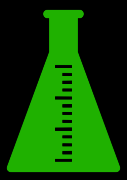
- **CORS (Cross-Origin Resource Sharing) and end…**



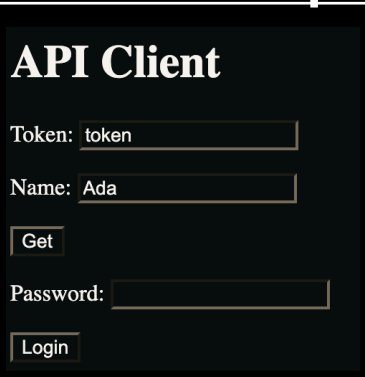
```
    <filter>
        <filter-name>cross-origin</filter-name>
        <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
        <init-param>
            <param-name>allowedOrigins</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>allowedMethods</param-name>
            <param-value>GET,POST</param-value>
        </init-param>
        <init-param>
            <param-name>allowedHeaders</param-name>
            <param-value>X-Requested-With,Content-Type,Accept,Origin,Authorization</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>cross-origin</filter-name>
        <filter-pattern>/*</filter-pattern>
    </filter-mapping>

</web-app>
```


- **With the env var, you can start jetty from other dirs.**
- **Restart jetty from the jetty-base dir:**
  - **java -jar $JETTY_HOME/start.jar**
  - **Load: http://localhost:8080/**
- **Load the web app at:**
  - **http://localhost:8080/PersonServlet/people/Kevin**

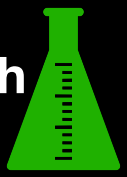
API Design: Java Server and Client



- **Create HTML client and place in webapps/root directory**
- **Load with:**
  - **http://localhost:8080/ApiClient.html**



API Design: Java Server and Client



- **DataStore is initialized with each start with preconfigured data:**

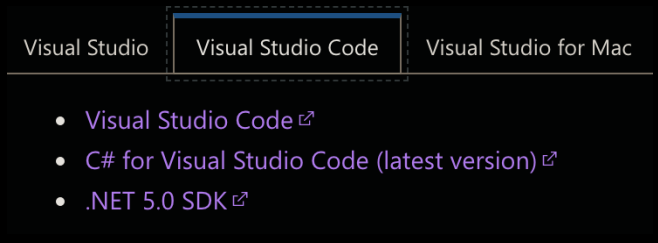
```
personMap.put("Ada", new Person("Ada", "Ada Lovelace was the first programmer.", 1815, "pwa"));
personMap.put("Kevin", new Person("Kevin", "Kevin is the author of HappyCoding.io.", 1986, "pwk"));
personMap.put("Stanley", new Person("Stanley", "Stanley is Kevin's cat.", 2007, "pws"));    }
```

- **Users can be fetched with GET without login**
- **Logged in users can Save changes to own account**
- **Logged in users can create new users**

C# .NET Server

API Design: C#.NET Server and Client

- **C# .NET tutorial**
- **https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio-code**



Java Standalone Server Using Spark

Java Standalone Server

- **Java JDK**
- **https://docs.oracle.com/en/java/javase/**
  - **Verify JAVA_HOME is set in env**

- **Maven**
- **https://maven.apache.org/download.cgi**
- **https://maven.apache.org/install.html**
  - **Verify maven bin dir is in PATH in env**

Java Standalone Server

- Create Java Maven Project
  - Create pom.xml
    - mvn archetype:generate -DgroupId=com.brainwashinc.demos.hellorestjava -DartifactId=hello-rest -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
  - Build Package
    - cd hello-rest; mvn clean package
  - Run
    - java -cp target/hello-rest-1.0-SNAPSHOT.jar com.brainwashinc.app.App

```
Bear-MBP:hello-rest bearc2020$ java -cp target/hello-rest-1.0-SNAPSHOT.jar com.brainwashinc.app.App
Hello World!
```


Java Standalone Server

- Spark
  - "A micro framework for create web applications"
  - Add to project: http://sparkjava.com/download
  - Add to pom.xml

```
<dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-core</artifactId>
    <version>2.9.3</version>
</dependency>
```

Java Standalone Server

- Spark
  - Update App class
    - Imports

```
import static spark.Spark.*;
import spark.Request;
import spark.Response;
import spark.Route;
```

    - Route

```
public class App
{
    public static void main( String[] args )
    {
        get("/hello", (req, res) -> "Hello World");
        // System.out.println( "Hello World!" );
    }
}
```

    - Build: mvn clean package (error: next slide)

Java Standalone Server

- Spark
  - Error

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-c
ompile) on project hello-rest: Compilation failure
[ERROR] /Users/Shared/Downloads/pomgen/hello-rest/src/main/java/com/brainwashinc/app/App.java:[16,34]
lambda expressions are not supported in -source 7
[ERROR]   (use -source 8 or higher to enable lambda expressions)
```

  - Update pom.xml maven to 1.8

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
```

Java Standalone Server

- **Spark**
  - **Run: e.g.,**
  - **mvn exec:java -Dexec.mainClass="com.brainwashinc.app.App"**

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

  - **Update pom.xml:**

```
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.21</version>
    </dependency>
```

  - **Rebuild, Rerun**

Java Standalone Server

- **Output with port: 4567**

```
[Thread-1] INFO org.eclipse.jetty.util.log - Logging initialized @1265ms to org.eclipse.jetty.util.log.Slf4jLog
[Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - == Spark has ignited
[Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - >> Listening on 0.0.0.0:4567
[Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.4.31.v20200723; built: 2020-07-23T17:57:36.812Z; git: 450ba27947e13e66baa8c
d1ce7e85a4461cacc1d; jvm 17+35-LTS-2724
[Thread-1] INFO org.eclipse.jetty.server.session - DefaultSessionIdManager workerName=node0
[Thread-1] INFO org.eclipse.jetty.server.session - No SessionScavenger set, using defaults
[Thread-1] INFO org.eclipse.jetty.server.session - node0 Scavenging every 600000ms
[Thread-1] INFO org.eclipse.jetty.server.AbstractConnector - Started ServerConnector@6becc6b5{HTTP/1.1, (http/1.1)}{0.0.0.0:4567}
[Thread-1] INFO org.eclipse.jetty.server.Server - Started @1354ms
```

- **Hit with browser or curl:**
  - **curl http://localhost:4567/hello**

Hello World

Java Standalone Server

- Change the main
  - Add a path for /user
    - Takes an id and returns the parameters

```
path("/user", () -> {
    get("/:id", (req, res) -> req.params());
});
```

- Rebuild; rerun
- Hit with browser or curl:
  - curl http://localhost:4567/user/123

```
{:id=123}
```


Java Standalone Server

- Change the main
  - Prefix that path with a v1 path

```
path("/v1", () -> {
    path("/user", () -> {
        get("/:id", (req, res) -> req.params());
    });
});
```

  - Rebuild; rerun
  - Hit http://localhost:4567/v1/user/123

```
{:id=123}
```

Java Standalone Server

- Add a before and after

```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"));
    path("/user", () -> {
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```

- Rebuild; rerun
- Hit http://localhost:4567/v1/user/123
- See log output

```
Received api call
Received api done
```


Java Standalone Server

- Add other methods

```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"))
    path("/user", () -> {
        post("", (req, res) -> "add user");
        put("/:id", (req, res) -> "replace user: " + req.params());
        delete("/:id", (req, res) -> "delete user: " + req.params()
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```

- Rebuild; rerun

Java Standalone Server

- **Of course you need actually functionality for the methods…**

```
path("/v1", () -> {
    before("/*", (q, a) -> System.out.print("Received api call\n"));
    path("/user", () -> {
        post("", (req, res) -> {
            // functionality goes here
            return "add user";
        });
        put("/:id", (req, res) -> "replace user: " + req.params());
        delete("/:id", (req, res) -> "delete user: " + req.params());
        get("/:id", (req, res) -> req.params());
    });
    after("/*", (q, a) -> System.out.print("Received api done\n"));
});
```


Java Standalone Server

- **Query those methods**
  - **curl -X POST http://localhost:4567/v1/user**
  - **curl -X PUT http://localhost:4567/v1/user/1**
  - **curl -X GET http://localhost:4567/v1/user/1**
  - **curl -X DELETE http://localhost:4567/v1/user/1**
- **Spark documentation:**
  - **http://sparkjava.com/documentation**

Python Standalone Server Using Flask

Python-Flask Standalone Server

- **Install python**
  - **https://www.python.org/downloads/**
- **Make a project directory (e.g., pyflaskrestful)**
- **Create and active an environment**
  - **python3 -m venv .venvrest**
  - **source .venvrest/bin/activate**
- **Install flash and flask-restful**
  - **pip3 install flask**
  - **pip3 install flask-restful**

Python-Flask Standalone Server

- Create an app.py file
  - Add imports
  - Create API object

```
# using flask_restful
from flask import Flask, jsonify, request
from flask_restful import Resource, Api

app = Flask(__name__)
# API object
api = Api(app)
```


Python-Flask Standalone Server

- Create a Resource and add it to the API
- Add the run statement

```
# class for a resource
class UserRoot(Resource):
    def get(self):
        return [{'userId': '123'}, {'userId': '1234'}]

# add the resource(s)
api.add_resource(UserRoot, '/user')

# driver
if __name__ == '__main__':
    app.run(debug = True)
```

Python-Flask Standalone Server

- **Start the process: python3 app.py**
  - **Uses 127.0.0.1:5000**

```
(.venvrest) Bear-MBP:pyflaskrestful bearc2020$ python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 255-296-312
```


Python-Flask Standalone Server

- **Hit the endpoint**
  - **curl http://127.0.0.1:5000/user**

```
Bear-MBP:javaserverSpark bearc2020$ curl http://127.0.0.1:5000/user
[
    {
        "userId": "123"
    },
    {
        "userId": "1234"
    }
]
```

Python-Flask Standalone Server

- Create a User level resource
  - Takes userId from path
- Add it below the existing UserRoot

```
class User(Resource):
    def get(self, userId):
        return {'userId': userId}

# add the resource(s)
api.add_resource(UserRoot, '/user')
api.add_resource(User, '/user/<userId>')
```


Python-Flask Standalone Server

- Hit the new endpoint
  - curl http://127.0.0.1:5000/user/123a

```
Bear-MBP:javaserverSpark bearc2020$ curl http://127.0.0.1:5000/user/123a
{
    "userId": "123a"
}
```

Python-Flask Standalone Server

- Create post method to UserRoot
  - Get the form data from the request

```
class UserRoot(Resource):
    def post(self):
        data = request.form['username']
        return {'username':data, 'userId':'123'}, 201
```

  - Hit the endpoint with data

```
curl -X POST http://127.0.0.1:5000/user -d "username=bear"
```

```
Bear-MBP:javaserverSpark bearc2020$ curl -X POST http://127.0.0.1:5000/user
-d "username=bear"
{
    "username": "bear",
    "userId": "123"
}
```

Python-Flask Standalone Server

- Change the post endpoint to expect a JSON body
  - Return data including body data

```
    def post(self):
        data = request.json
        return {"username":data["username"], "userId":"123"}
```

Python-Flask Standalone Server

- **Hit the endpoint with JSON data and content-type header**
- **curl -X POST http://127.0.0.1:5000/user -d '{"username":"bear","email":"test@wx.com"}' -H "Content-Type: application/json"**

```
Bear-MBP:javaserverSpark bearc2020$ curl -X POST http://127.0.0.1:5000/user -d
'{"username":"bear","email":"test@wx.com"}' -H "Content-Type: application/json"
{
    "username": "bear",
    "userId": "123"
}
```


Python-Flask Standalone Server

Final Full Code

-

```
# using flask_restful
from flask import Flask, jsonify, request
from flask_restful import Resource, Api

app = Flask(__name__)
# API object
api = Api(app)

# class for a resource
class UserRoot(Resource):
    def post(self):
        data = request.json
#        return data
        return {"username":data["username"], "userId":"123"}
#        data = request.form['username']
#        return {'username':data, 'userId':'123'}, 201

    def get(self):
        return [{'userId': '123'}, {'userId': '1234'}]

class User(Resource):
    def get(self, userId):
        return {'userId': userId}

# add the resource(s)
api.add_resource(UserRoot, '/user')
api.add_resource(User, '/user/<userId>')

# driver
if __name__ == '__main__':
    app.run(debug = True)
```

RESTful API Design and Development

The End