



RESTful API Design and Development

<https://www.accelebrate.com>
(877) 849-1850 ♦ info@accelebrate.com

Customized Technical Training



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

Visit our website for a complete list of courses!

| | | | |
|---------------------------------|--|--|---|
| Adobe & Articulate | Power BI PivotTable and PowerPivot PostgreSQL SQL Server Vertica Architecture & SQL | Flutter HTML5 iOS/Swift Development JavaScript MEAN Stack Mobile Web Development DevOps, CI/CD & Agile | SharePoint Power Automate & Flow SharePoint Administrator SharePoint Developer SharePoint End User SharePoint Online SharePoint Site Owner |
| AWS, Azure, & Cloud | Agile Ansible Chef DEI Docker Git Gradle Build System IT Leadership | React & Redux Svelte Swift Xamarin Vue | SQL Server Azure SQL Data Warehouse Business Intelligence Performance Tuning SQL Server Administration SQL Server Development SSAS, SSIS, SSRS Transact-SQL |
| Big Data | ITIL Jenkins Jira & Confluence Kubernetes Linux Microservices OpenShift Red Hat Software Design | 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 | Teleconferencing Tools Adobe Connect GoToMeeting Microsoft Teams WebEx Zoom |
| Data Science and RPA | Java Apache Maven Apache Tomcat Groovy and Grails Hibernate Java & Web App Security JavaFX JBoss Oracle WebLogic Scala | 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 | Web/Application Server Apache httpd Apache Tomcat IIS JBoss Nginx Oracle WebLogic |
| Database & Reporting | Selenium & Cucumber Spring Boot Spring Framework JS, HTML5, & Mobile Angular Apache Cordova CSS D3.js | 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 | 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.

Understanding APIs in Software Delivery: Introduction



Bear Cahill
BrainwashInc.com

Freelance Mobile Developer

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

1

Understanding APIs in Software Delivery: Introduction

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

The screenshot shows the Insomnia API client. At the top, there's a navigation bar with the Insomnia logo, a star rating of 18,478, and links for Products, Docs, Pricing, Plugins, Login, and Get Started for Free. Below the navigation is a dark header with the text "Download Insomnia". Underneath, it says "Start building, designing, testing better APIs through spec-first development driven by an APIOps CI/CD pipelines." A blue button labeled "Download Insomnia for MacOS" is visible. At the bottom left, there's a "What's New? Changelog" link and a note about the latest OS X download. On the right side, the main interface shows a "DOCUMENT" tab with a dropdown for "No Environment". It lists several endpoints under categories like "pet", "store", and "user". A "POST" request is selected with the URL "/new_user", and the JSON body is displayed as:

```
{  
  "id": 6,  
  "category": {  
    "id": 6,  
    "name": "string"  
  },  
  "name": "doggie",  
  "photourl": {  
    "string"  
  },  
  "tags": [  
    {  
      "id": 6,  
      "name": "string"  
    }  
  ],  
  "status": "string"  
}
```

Understanding APIs in Software Delivery: Introduction

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



3

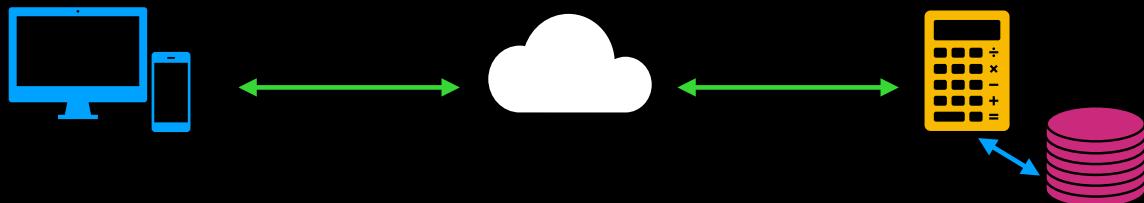
Understanding APIs in Software Delivery: Introduction

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



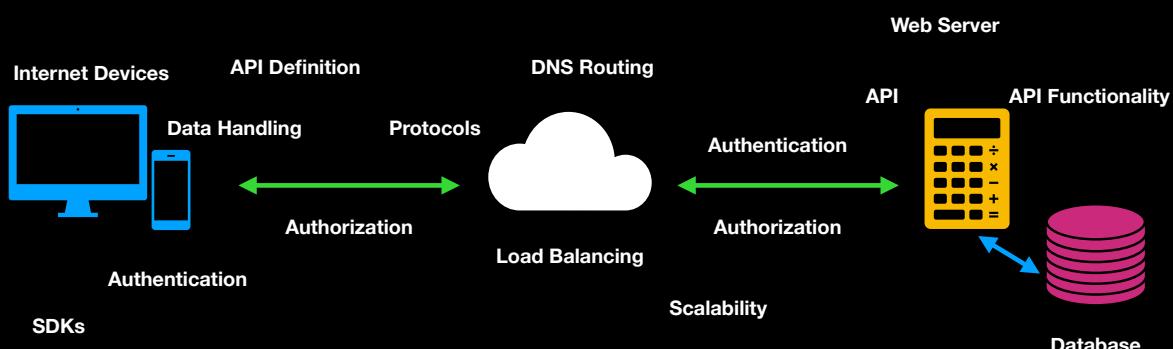
4

API Design: Introduction



5

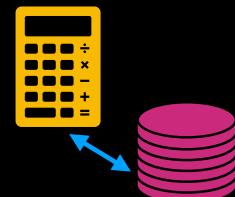
API Design: Introduction



6

API Design: Introduction

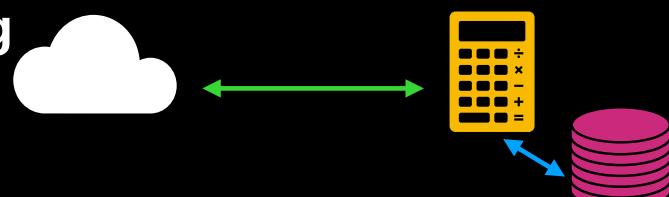
- **API Backend**
 - **Servers (e.g., Apache, Tomcat)**
 - **Web Server Apps**
 - **Processes API Requests**
 - **Functionality**
 - **Java, C#, Python, PHP, etc.**
 - **Data**
 - **Database (e.g., Oracle, MySQL)**



7

API Design: Introduction

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



8

API Design: Introduction

- **API Client**
 - **Device on Internet**
 - **Understands API**
 - **Endpoints (Nouns, Verbs)**
 - **Data formats (JSON)**
 - **Login/Authentication**
 - **e.g., Token Header**



9

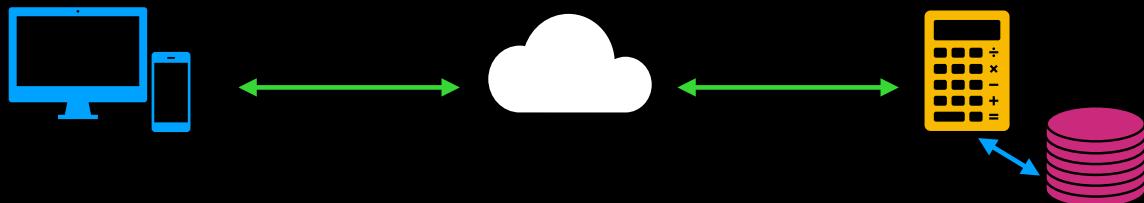
API Design: Introduction

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



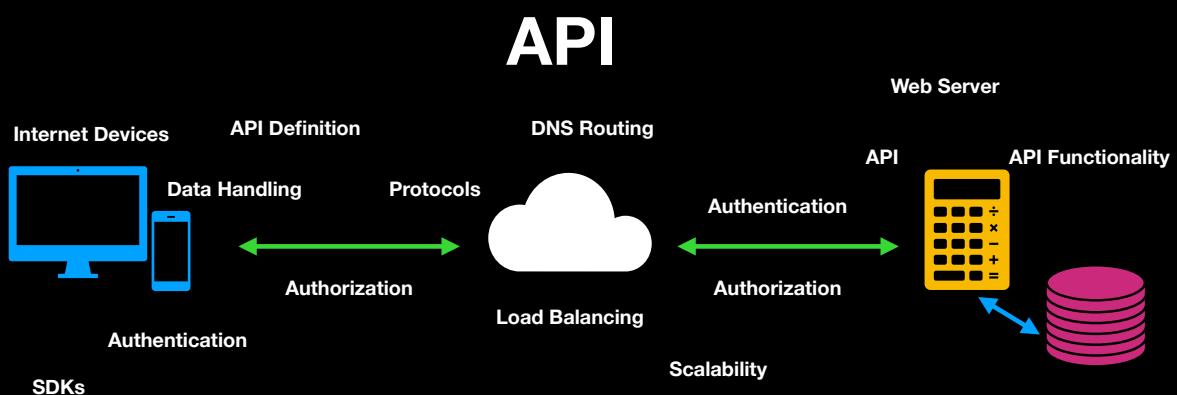
10

API Design: Introduction



11

API Design: Introduction

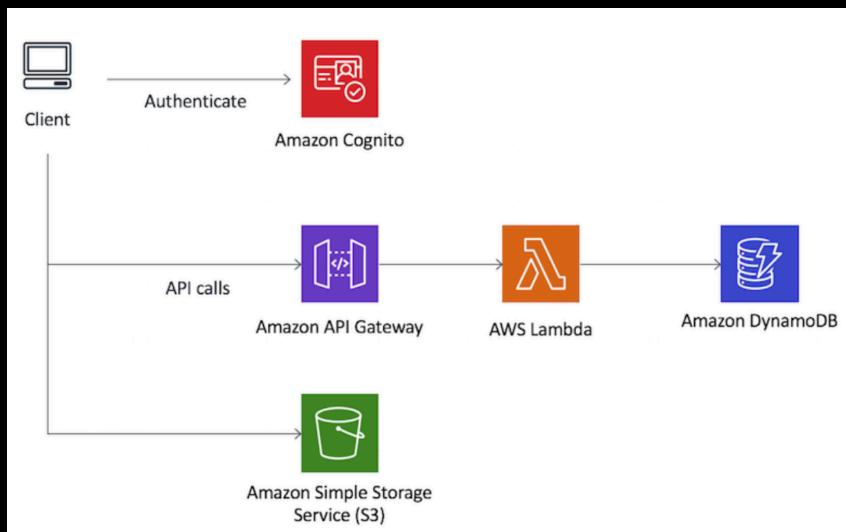


OpenAPI Spec

12

API Design: Introduction

- **AWS**



13

API Design: Introduction



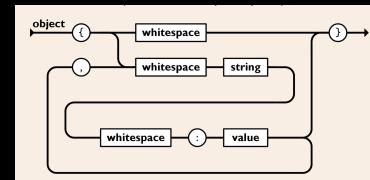
- **Past:**
 - **API Wild West**
 - **URL/Query String, Form Data, etc.**
 - **SOAP/WSDL**
 - **XML, text, etc.**
 - **Everything was proprietary, complex**
 - **Needed a Standard**

14

API Design: Introduction

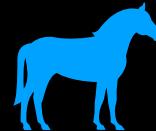


- **2000:**
 - Roy Fielding's doctoral dissertation on REST (REpresentational State Transfer)
- **2001:**
 - JSON by Douglas Crockford
 - JavaScript Object Notation
 - json.org



15

API Design: Introduction



- **Early 2000s - Money to be made**
 - Access data
 - Useful functionality, storage
 - eBay, Flickr, Amazon, Twitter, etc.
 - The standard is key

16



- REST (or other) API is expected now
- Libraries/frameworks/SDKs abound
- Variations cause problems
- Downside: better solutions will suffer from inertia



- 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.

```
{  
  user(id: 4802170) {  
    id  
    name  
    isViewerFriend  
    profilePicture(size: 50) {  
      url  
      width  
      height  
    }  
    friendConnection(first: 5) {  
      totalCount  
      friends {  
        id  
        name  
      }  
    }  
  }  
}  
  
{  
  "data": {  
    "user": {  
      "id": "4802170",  
      "name": "Zed Gee",  
      "isViewerFriend":  
        "profilePicture":  
          "url": "cdn://  
            "width": 50,  
            "height": 50  
        },  
      "friendConnection":  
        "totalCount":  
          "friends": {  
            "id": "305",  
            "name": "S",  
            "id": "310",  
            "name": "B"  
          }  
    }  
}
```

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**

19

OpenAPI Object
This is the root document object of the API.

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 |

API Design: Documenting



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

Components Object

Holds a set of reusable objects for different aspects of the OAS. All objects define 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] |

20

API Design: Introduction

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

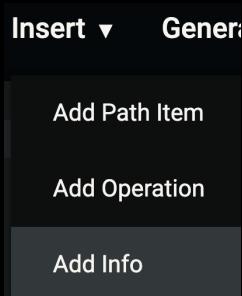
The screenshot shows the Swagger Editor interface with a pink Tink logo in the top right corner. The title bar says "Swagger Editor™ Supported by SMARTBEAR". Below it, a banner indicates "openapi: 3.0.1". The main area is titled "Errors" and contains two error messages:

- Structural error at**
should have required property 'info'
missingProperty: info
[Jump to line 0](#)
- Structural error at**
should have required property 'paths'
missingProperty: paths
[Jump to line 0](#)

21

API Design: Introduction

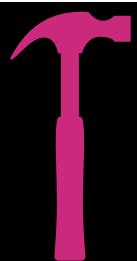
- **Editor**
 - **Insert > Info**
 - **Enter text in fields**
 - **License/Contact optional**
- Click 'Add Info'**



Add Info to Document

| |
|--|
| Title * REQUIRED. The title of the application. <input type="text" value="My API"/> |
| Description A short description of the application. Com <input type="text" value="Best API ever"/> |
| Version * REQUIRED. The version of the OpenAPI d <input type="text" value="1.0"/> |
| Terms of Service A URL to the Terms of Service for the API. I <input type="text" value="https://www.example.com/"/> |

22



- **Text inserted**

```

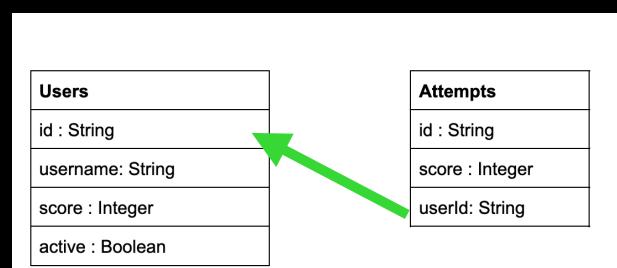
1  openapi: 3.0.1
2  info:
3    title: My API
4    version: '1.0'
5    description: Best API ever
6    termsOfService: https://www.example.com/
7    contact:
8      email: bear@brainwashinc.com
9      name: Bear Cahill
10     url: http://www.example.com/contact
11    license:
12      name: My License
13      url: http://www.example.com/license

```

- **End of lab**



- **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 ']'
```

25

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 |

26

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 ']'
```



27

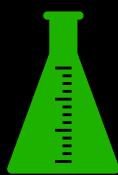
API Design: JSON



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

28

API Design: JSON

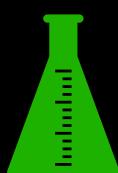


- **Lab**
 - <https://jsonlint.com/>
 - **Create an array of Elements**
 - **Each element has 4 members:**
 - **id of type String**
 - **username of type String**
 - **score of type Integer**
 - **active of type Boolean**
 - **See next slide for hints**

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

29

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**

30

API Design: JSON



- **Lab**
- **End of 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  
}]
```

31

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**

32



- **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**

33



- **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: { }**

34

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"  
  }]  
}
```

35

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"  
  }]  
}
```

36

API Design: Design

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

37

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

- **Lab: GET** API Design: Design
- **<https://my-json-server.typicode.com/typicode/demo>**
- **View db.json file**
- **Via browser:**
 - **GET posts, single post, comments, single comment, comments for 1 post**



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

How to

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

No registration. Nothing to install.



Sponsor

Share

Fork

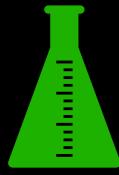
demo

by typicode

Available resources

• posts 3

API Design: Design

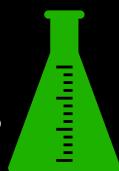


- **Lab: GET - Different db.json**
- **<https://my-json-server.typicode.com/bearc0025/api>**
- **View db.json file**
- **Via browser:**
 - **GET users, 1 user, attempts, 1 attempt, etc.**

39

```
{
  "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)**

Insert ▾ Generate Server ▾

Add Path Item
Add Operation
Add Info
Add External Documentation
Add Tag Declarations
Add Tags To Operation
Add Servers

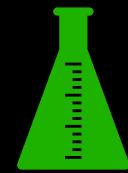
URL *
REQUIRED. A URL to the target host. This URL supports Server relative to the location where the OpenAPI document is being served {brackets}.

Description
An optional string describing the host designated by the URL. Consider

40

API Design: Design

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



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

Servers

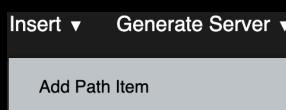
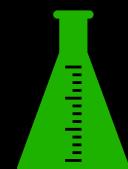
https://my-json-server.typicode.com/bearc0025/api - User API ▾

- **End of lab**

41

API Design: Design

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



Add Path

Path *
REQUIRED. The path to add.

Summary
Enter a summary of the path.

Description
An optional, string description

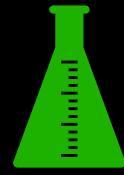
Creates:

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

42

API Design: Design

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



Creates:

Insert ▾ Generate Server

Add Path Item

Add Operation

Add Operation to Document

Path *
REQUIRED. The path to add the operation to.
`/users`

Operation *
REQUIRED. Select an operation.
`get`

Summary
Add a short summary of what the operation does.
`fetch users`

Description
A verbose explanation of the operation behavior. Comments about how the operation is used.
`read users from server`

Operation ID
Unique string used to identify the operation. The id must be unique across all operations in the API.
`fetchUsers`

Tags
A list of tags for API documentation control. Tags can be used to filter and group operations.

Tag *
REQUIRED. The name of the tag.
`users`

```
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
```

43

API Design: Design

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



users

GET /users fetch users

GET /users fetch users

read users from server

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' '\n"https://my-json-server.typicode.com/search0025/api/users"\n--accept "/*"\n'
```

Request URL

`https://my-json-server.typicode.com/search0025/api/users`

Server response

Code Details

200

Response body

```
[{"id": "1234abc", "username": "beast", "score": 500, "active": true}, {"id": "xyz123", "username": "sheesh", "score": 350, "active": false}, {"id": "werewy", "username": "jed", "score": 350, "active": true}]
```

Download

- **End of lab**

44

API Design: Design

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

45

```
{  
  "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**

46

```
{  
  "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"  
  }]  
}
```

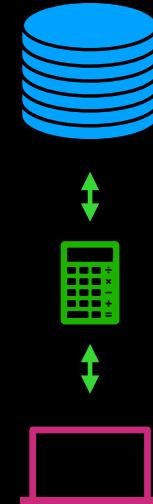
API Design: Design

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

47

```
{  
  "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



48

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
 - Scalability
 - Reliability
 - Etc.



49

API Design: Design Concepts

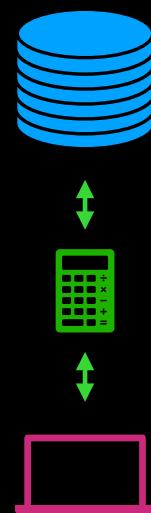
- **Separation of Concerns**
 - **Gathering Data**
 - When requested by client
 - **Very important on the server**
 - Good database design
 - Cache if possible
 - Efficient queries



50

API Design: Design Concepts

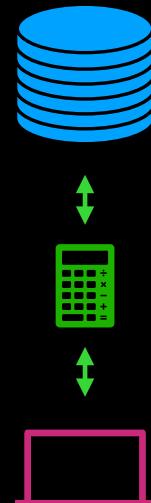
- **Separation of Concerns**
 - **Formatting Data**
 - When sending/returning to client
 - **Important to client**
 - Directed by server (one-for-all)



51

API Design: Design Concepts

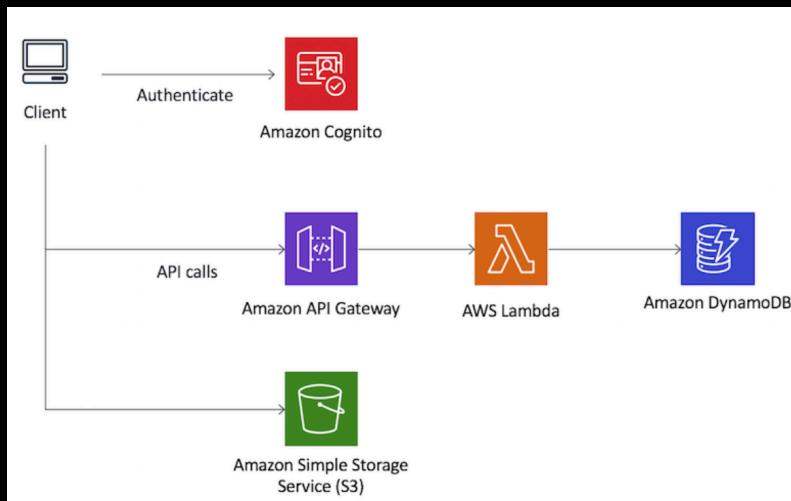
- **Separation of Concerns**
 - **Delivering Data**
 - How it is sent: protocol, etc.
 - **Important to client**
 - **Directed by server**
 - Usually the same for all clients



52

API Design: Design Concepts

- **AWS - Separation of Concerns**



53

API Design: Design



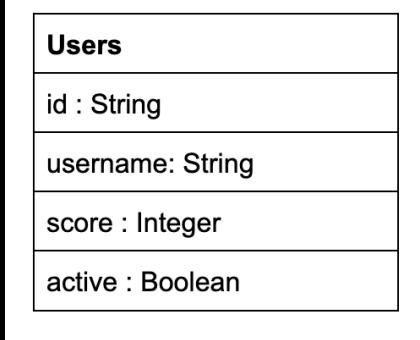
- **Operations: Responses**
 - **Default Response**
 - **Specified by Status Code (e.g., 200, 404, etc.)**
- **Includes:**
 - **Description**
 - **Content**
 - **Content Type**
 - **Schema**

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

54

API Design: Design

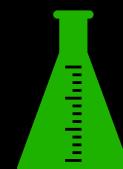
- **For all 200-299 responses**
- **application/json content**
- **Schema is an object**
- **With properties**



```
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
```

55

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



```
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
```

Responses

| Code | Description |
|------|------------------|
| 2XX | successful fetch |

Media type

application/json

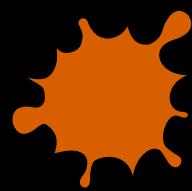
Controls Accept header.

Example Value | Schema

```
{ "id": "abc123", "username": "bear", "score": 55, "active": true }
```

- **End of lab**

56



- **Errors**
 - **HTTP**
 - **Response Codes**
 - **1xx: Informational**
 - **2xx: Success**
 - **3xx: Redirection**
 - **4xx: Client Error**
 - **5xx: Server Error**
 - **<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>**



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

API Design: Design

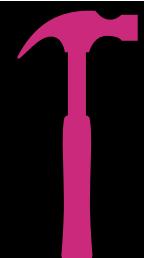
- **Lab**
- **If you have a git account. Create a public repo...**
- **<https://my-json-server.typicode.com/>**
- **Demo data @ <https://github.com/bearc0025/api/blob/main/db.json>**



How to

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

API Design: Design Concepts



- **Now that we have two...**
- **Add Another Server**
- **[editor.swagger.io](#)**
- **Insert > Add Servers**

The screenshot shows the Swagger Editor interface with the 'Add Servers' dialog open. The dialog has a title bar 'Add Servers' and a sub-section 'Server'. It defines a 'Server' as 'An object representing a Server'. The main form contains fields for 'URL *' (with the value 'https://my-json-server.typicode.com/bearc0025/api') and 'Description' (with the value 'My GIT server'). A 'Server Variables' section is also present.

- **Add your server or mine:**
 - <https://my-json-server.typicode.com/bearc0025/api>

API Design: Richardson Maturity Model



61

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)**



62

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



63

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.**



64

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.**



65

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"  
        }  
    ]  
}
```

66

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"  
        }  
    ]  
}
```



67

API Design: Components

- Many sections for reuse
- Note: schemas
 - Can define schemas
 - Reference in Operations
- Also, responses
 - Reference schemas
 - Reference in Operations
- <https://swagger.io/specification/>

| 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. |



68

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**

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

```
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**

69



API Design: Design

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

The screenshot shows the PokeAPI homepage with the title "PokéAPI" and subtitle "The RESTful Pokémon API". It mentions serving over 60,000,000 API calls each month. A "Check out the docs!" button is present. Below, a "Try it now!" button is shown with a URL input field containing "https://pokeapi.co/api/v2/pokemon/ditto". A "Submit" button is next to it. The page then displays the "Resource for ditto" with the following JSON-like data:

```
* abilities: [] 2 items  
  * 0: {} 3 keys  
    * ability: {} 2 keys  
      name: "flame" url: "https://pokeapi.co/api/v2/ability/7/"  
    is_hidden: false slot: 1
```

70

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

The screenshot shows the SWAPI (The Star Wars API) website. At the top, there's a green flask icon and the text "SWAPI" followed by "The Star Wars API" and "(what happened to swapi.co?)". Below this, a banner says "All the Star Wars data you've ever wanted: Planets, Spaceships, Vehicles, People, Films and Species From all SEVEN Star Wars films Now with The Force Awakens data!". A "Try it now!" button is visible. In the middle, there's a search bar with the URL "https://swapi.dev/api/people/1/" and a "request" button. The result section shows a JSON response for Luke Skywalker:

```
{  
  "name": "Luke Skywalker",  
  "height": "172",  
  "mass": "77",  
  "hair_color": "blond",  
  "skin_color": "fair",  
  "eye_color": "brown",  
  "birth_year": "19BBY",  
  "gender": "male",  
  "homeworld": "https://swapi.dev/api/planets/1/",  
  "species": "https://swapi.dev/api/species/1/",  
  "films": "https://swapi.dev/api/films/1/",  
  "vehicles": "https://swapi.dev/api/vehicles/1/",  
  "starships": "https://swapi.dev/api/starships/1/",  
  "created": "2014-12-09T13:50:51.651Z",  
  "edited": "2014-12-09T13:50:51.651Z",  
  "url": "https://swapi.dev/api/people/1/"}
```

71

API Design: Testing and Development

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



The screenshot shows the Postman application interface. At the top, there's a navigation bar with "Tests", "Watch", "0", "Fork", "0", "Run", "Save", "Share", and "..." buttons. The "Tests" tab is selected. Below the navigation bar, there are tabs for "Authorization", "Pre-request Script", "Tests" (which is green), and "Variables". A note says "These tests will execute after every request in this collection. [Learn more about Postman's execution order.](#)"

In the "Tests" section, there's a code editor with the following JavaScript:

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Below the code editor, a note says "Test scripts are written in JavaScript, and are run after the response is received." and a link "Learn more about tests scripts". There are also "SNIPPETS" and "Get an environment variable" buttons.

To the right, there's a "Tests" panel titled "Tests: No Environment, just now". It shows a table with "All Tests", "Passed (1)", and "Failed (4)". The "Iteration 1" section lists four failed tests:

- GET get Post By Id [baseURL]/posts/17880243 | Fail Status code is 200 | Assertion Error: expected 200 to be 200
- DELETE /posts/{id} [baseURL]/posts/-17880243 | Fail Status code is 200 | Assertion Error: expected 200 to be 200
- PATCH update post [baseURL]/posts/-17880243 | Fail Status code is 200 | Assertion Error: expected 200 to be 200
- GET fetch posts [baseURL]/posts | /posts / fetch | Pass Status code is 200

At the bottom, there are "POST create post" and "PUT update post" entries with their respective status codes and error messages.

72

API Design: Testing and Development



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

A screenshot of the Prism API testing tool interface. At the top, there are tabs for 'Form', 'Code', 'Preview', 'Mocks', and '14 Problems'. Below the tabs, there are two tabs: 'Routes' (selected) and 'Logs'. The 'Routes' tab displays a table of API endpoints with columns for Method, Path, and Mock URL. The table contains the following data:

| Method | Path | Mock URL |
|--------|---------------|------------------------------------|
| GET | /pets | http://127.0.0.1:3106/pets |
| POST | /pets | http://127.0.0.1:3106/pets |
| GET | /pets/{petId} | http://127.0.0.1:3106/pets/{petId} |
| GET | /employee | http://127.0.0.1:3106/employee |
| POST | /employee | http://127.0.0.1:3106/employee |
| PUT | /employee | http://127.0.0.1:3106/employee |
| PATCH | /employee | http://127.0.0.1:3106/employee |
| DELETE | /employee | http://127.0.0.1:3106/employee |

73

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**

74

API Design: Dev Standards



75

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**



76



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

77



- **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**

78

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**



79

API Design: Dev Standards

- **Filtering/Sorting/Search/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



80

- **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**



81

- **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
```

82

- **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
```

83

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>**



84

API Design: Contracts

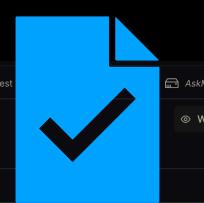
- **OpenAPI Documentation**
- **<https://swagger.io/specification/>**
- **Various sections and types**

| 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 <code>openapi</code> field SHOULD be used by tooling specifications and clients to interpret the OpenAPI document. This is <i>not</i> related to the API <code>info.version</code> 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 <code>servers</code> property is not provided, or is an empty array, the default value would be a <code>Server Object</code> with a <code>url</code> value of <code>/</code> . |
| 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 <code>({})</code> 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 <code>Operation Object</code> 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. |

85

API Design: Contracts

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

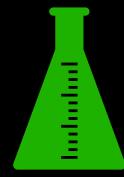


```

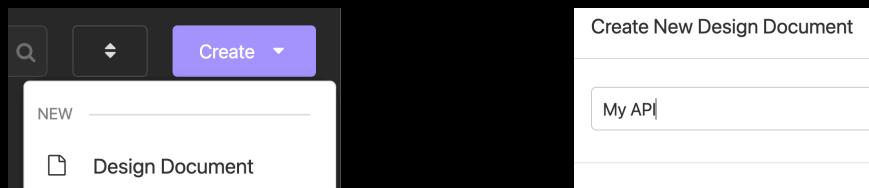
1  {
2    "openapi": "3.0.0",
3    "info": {
4      "version": "1.0.0",
5      "title": "AskIT",
6      "license": {
7        "name": "MIT"
8      },
9    },
10   "servers": [
11     {
12       "url": "https://my-json-server.typicode.com/bear"
13     }
14   ],
15   "paths": {
16     "/posts": {
17       "get": {
18         "summary": "Details about a post",
19         "operationId": "listPost",
20         "tags": [
21           "post"
22         ],
23         "parameters": [
24           {
25             "name": "id",
26             "in": "query"
27           }
28         ]
29       }
30     }
31   }
32 }
```

86

API Design: Insomnia



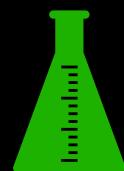
- **Create API in Insomnia**
 - **Open Insomnia**
 - **Click Create > Design Document**
 - **Give it a name and click Create**



87

API Design: Insomnia

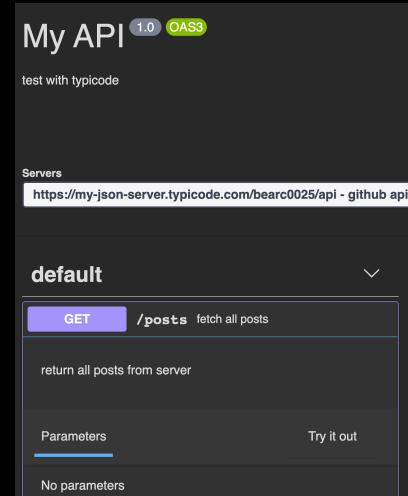
- **Paste your texts from the Swagger editor into the Insomnia editor**

A screenshot of the Insomnia application window. The title bar says 'Dashboard / My API'. Below it, there are tabs for 'DESIGN' and 'DEBUG', with 'DESIGN' being active. On the left, there are three sections: 'INFO', 'SERVERS', and 'PATHS'. The 'PATHS' section contains a large block of JSON code representing a Swagger specification. The code includes definitions for 'openapi', 'info', 'servers', and a 'posts' path with a 'get' operation.

88

API Design: Insomnia

- **Test on the right**



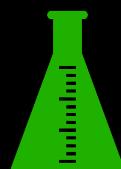
The screenshot shows the Insomnia API testing tool interface. At the top, it says "My API 1.0 OAS3" and "test with typicode". On the right, there's a green flask icon. Below that, under "Servers", is the URL "https://my-json-server.typicode.com/bearc0025/api - github api". A dropdown menu is open, showing "default". Under "default", there's a list of API endpoints. The first endpoint is a "GET /posts" request, which is highlighted with a purple button. The description for this endpoint is "fetch all posts" and the response is "return all posts from server". Below the endpoint, there are sections for "Parameters" and "Try it out". The "Parameters" section has a blue underline and the text "No parameters".

- **End of lab**

89

API Design: RESTful API Basics

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



90

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>**

Getting started

Install JSON Server

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

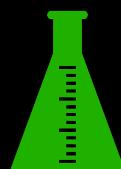
Create a db.json file with some data

91

```
{
  "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": "al",
      "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>**

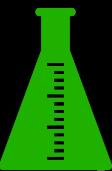


```
Bear-MBP:json-server bearc2022$ json-server --watch db.json
  \{^_^\}/ hi!
  Loading db.json
  Done
  Resources
  http://localhost:3000/users
  http://localhost:3000/attempts
  Home
  http://localhost:3000
```

92

```
localhost:3000/users/1234abc
{
  "id": "1234abc",
  "username": "bear",
  "score": 55,
  "active": true
}
```

API Design: RESTful API Basics

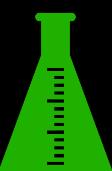


- Lab
- <http://localhost:3000/users/qwerty/attempts>

```
[  
  {  
    "id": "a1",  
    "score": 5,  
    "userId": "qwerty"  
  },  
  {  
    "id": "999",  
    "score": 8,  
    "userId": "qwerty"  
  }]
```

93

API Design: RESTful API Basics



Typicode - fake server -> typicode/demo (repo)

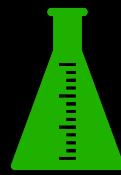
Typicode - fake server -> myaccount/myrepo (repo)

Local typicode server
-> Local db.json

Browser, OpenAPI editor, curl, Insomnia, Postman, etc.

94

API Design: RESTful API Basics



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

The screenshot shows the 'Servers' configuration panel in the Swagger UI. On the left, there is a sidebar with options like 'Insert', 'Generate', 'Add Path Item', 'Add Operation', etc. The main area displays the 'servers:' configuration:

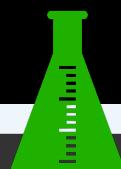
```
servers:
  - url: http://localhost:3000
    variables: {}
    description: Local User API
  - url: https://my-json-server.typicode.com/bearc0025/api
    variables: {}
    description: User API
```

Below this, a 'Servers' section lists two entries:

- ✓ http://localhost:3000 - Local User API
- https://my-json-server.typicode.com/bearc0025/api - User API

95

API Design: RESTful API Basics



- **Lab**
- **Test against local server**

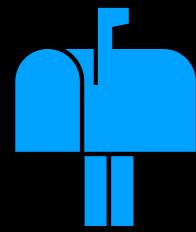
The screenshot shows a successful API call response from the 'users' endpoint. The request URL is 'http://localhost:3000/users'. The response code is 200, and the response body is a JSON array of user objects:

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

There are 'Code' and 'Details' tabs at the bottom of the response panel.

- **End of lab**

96



- **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
}
```

97



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

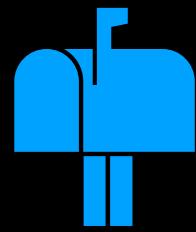
Request Body Object
Describes a single request body.

Fixed Fields

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

98

API Design: RESTful API Basics



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

Add Operation to Document

Path *
REQUIRED. The path to add the operation to.

Operation *
REQUIRED. Select an operation.

Summary
Add a short summary of what the operation does.

Description
A verbose explanation of the operation behavior. Consider:

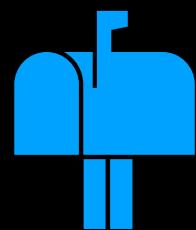
Operation ID
Unique string used to identify the operation. The id must be unique across all operations. Consider:

Tags
A list of tags for API documentation control. Tags can be used for dynamic generation of client libraries.

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

99

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)**

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

100

API Design: RESTful API Basics

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

```
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
```

101

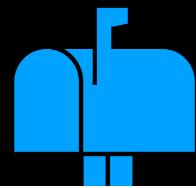
API Design: RESTful API Basics

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

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

102

API Design: RESTful API Basics



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

- **End of lab**

103

The screenshot shows a REST API endpoint for creating a user. The method is POST, the URL is /users, and the operation is 'create user'. The description is 'store user on server'. The 'Parameters' section indicates 'No parameters'. The 'Request body' is marked as required and has a type of application/json. An example request body is shown as a JSON object:

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

API Design: RESTful API Basics



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

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

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

- **End of lab.**

104

API Design



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

```
Dog: # "Dog" is a value for the pet_type property
      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]
```

```
requestBody:
  description: A JSON object containing pet information
  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**

```
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

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

- End of lab

107

API Design



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

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

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

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

- End of lab

108

API Design



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

```
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'
```

```
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
```

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

- **End of lab**

109

API Design



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

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

```
/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**

110



- **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**

111



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

112

API Design



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

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

Server Variables
A map between a variable name and its value. The value is used for substitution in URLs.

| | |
|--|---|
| Variable Name * | Default * |
| The name of the server variable. | REQUIRED. The default value is used if no other value is provided. An alternate value is resolved by substituting the variable name with its value. This value MUST be present. |
| scheme | https |
| Enum | |
| An enumeration of strings. Values must be unique and are from a limited set. | |
| Enum Value | http |
| Enum Value | https |

113

API Design



- **Server Variables**

```
- 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'
```

Servers

{scheme}://www.example.com:{port}/{version} - variables

Computed URL: <https://www.example.com:8080/v2>

Server variables

| | |
|---------|-------|
| scheme | https |
| version | v2 |
| port | 8080 |

114



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

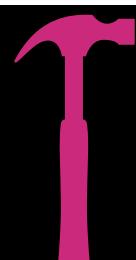


- **OpenAPI: Parameters**
 - **in:**
 - **path - URL**
 - **query - Query String**
 - **header - request header**
 - **cookie - Cookie**

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

117

API Design



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

Add Path

Insert ▾ Generat

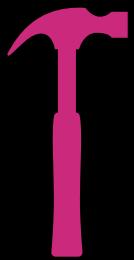
Add Path Item

| | |
|----------------------------|---|
| Path * | REQUIRED. The path to add. |
| /users/{userId} | |
| Summary | Enter a summary of the path. |
| specific user operation | |
| Description | An optional, string description, intended to help document the API. |
| fetch, update, delete user | |

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

118

API Design



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

Insert ▾ Generate
Add Path Item
Add Operation

Add Operation to Document

Path *
REQUIRED. The path to add the operation to.
`/users/{userId}`

Operation *
REQUIRED. Select an operation.
`get`

Summary
Add a short summary of what the operation does.
`fetch user`

Description
A verbose explanation of the operation behavior. Consider adding examples.
`fetch user by id`

Operation ID
Unique string used to identify the operation. The id MUST be unique across all operations in the API to uniquely identify an operation, therefore it is recommended to use the operation name.
`fetchUser`

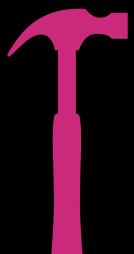
Tags
A list of tags for API documentation control. Tags can be used for dynamic generation of API documentation.
`users`

Tag *
REQUIRED. The name of the tag.
`users`

```
/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
```

119

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



```
/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
```

Request URL
`http://localhost:3000/users/1234abc`

Server response

| Code | Details |
|------|---|
| 200 | <p>Response body</p> <pre>{ "id": "1234abc", "username": "bear", "score": 55, "active": true }</pre> <p>Edit Download</p> |

120



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

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

121



- **OpenAPI: Parameters**
 - **header - request header**
 - **e.g., X-Request-ID:**
77e1c83b-7bb0-437b-bc50-a7a58e5660ac

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

122

API Design



- **OpenAPI: Parameters**
 - **cookie - Cookie**
 - **e.g., Cookie: debug=0;**
csrftoken=BUSe35dohU3O1MZvDCUOJ

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

API Design



- **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/>**

```
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**

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

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

125

API Design



- **OpenAPI: Parameters**
 - **May have an example(s)**
 - **May be an enum**

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

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

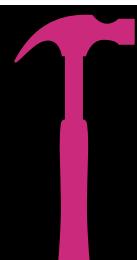
126



- **OpenAPI: Parameters**
 - **Parameters shared by multiple methods may be defined on the path level.**
 - **e.g., Used by GET and DELETE**

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

127



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

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

128

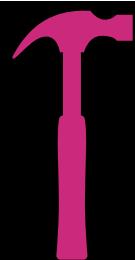


- **OpenAPI: Parameters**
- **May be in components**
- **Referenced with \$ref**

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

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

129



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

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

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

- **End of lab**

130



- **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)

131



- **Path Parameters**
 - **Styles:**
 - **simple (default)**
 - **label**
 - **matrix**
 - **Explode**
 - **true**
 - **false (default)**

132



- **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

```
/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
```

133



- **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

```
/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
```

134

API Design



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

```
-json-
picode.com/bearc0025/api/rates/{item},text,this%20post
```

135

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

API Design



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

```
.com/bearc0025/api/rates/{item*},text=this%20post
```

136

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

API Design



- **Styles: label - primitive/array**
 - **Dot-prefixed (aka Label Expansion)**
 - **{.item}**

Request URL

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

- **{.item*} for array**

URL

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

137

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

API Design



- **Styles: label - object**
 - **explode: false**

URL

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

- **explode: true**

my-json-

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

138

```
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
```



- **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
```

139



- **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
```

140



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

141

- **Query Parameter Styles**
 - **form**

Request URL

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

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

Array

- **Explode false**

Request URL

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

- **Explode true**

Request URL

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

142

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

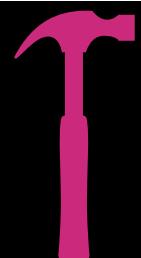
Cookie parameters always use the `form` style. An optional `explode` keyword controls the array and object serialization. Given the cookie named `id`, the cookie value is serialized as follows:

| 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

145

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'
```

146

API Design

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

Add Operation to Document

Path *
REQUIRED. The path to add the operation to.
`/users/{userId}`

Operation *
REQUIRED. Select an operation.
`delete`

Summary
Add a short summary of what the operation does.
`delete a user`

Description
A verbose explanation of the operation behavior. Commonly used to describe the effect of the operation.
`remove user from server`

Operation ID
Unique string used to identify the operation. The id MUST be unique across all operations. It is recommended to use the operationId to uniquely identify an operation, therefore, the value of operationId is `deleteUser`.

Tags
A list of tags for API documentation control. Tags can be used to group operations together, so that they can be displayed together.

Tag *
REQUIRED. The name of the tag.
`users`

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

- End of Lab

API Design

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

Path *
REQUIRED. The path to add the operation to.
`/users/{userId}`

Operation *
REQUIRED. Select an operation.
`patch`

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

API Design



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

The screenshot shows the Swagger UI interface for a PATCH request. On the left, a "Response body" panel displays a JSON object with fields: "username": "bear", "score": 55, "active": true, and "id": "i1HYeyQ". In the center, a "Request body" panel shows a JSON object with the same fields, but the "active" field is explicitly set to false. A green arrow points from the "active: true" value in the response body to the "active: false" value in the request body. On the right, another "Response body" panel shows the updated JSON object where "active" is now false.

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

149

API Design: Design First



150

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**



151

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**



152

API Design: Code First



153

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**



154

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)**



155

API Design: Automated Testing



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

156

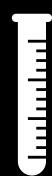
API Design: Automated Testing



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

157

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**

158

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."**



Chai Assertion Library

159

API Design: Automated Testing



- **Insomnia**

- **<https://www.chaijs.com/api/bdd/>**

BDD

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

API Reference

Language Chains

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

Chains

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

`.lengthOf(n[, msg])`

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

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

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

160

API Design: Automated Testing



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

The screenshot shows the Insomnia interface. On the left, under 'New Suite', there's a tree view with a root node 'Returns 200' and a child node 'Fetch All Users'. The code for the 'Fetch All Users' test is displayed:

```
1 const response1 = await insomnia.send();
2 const body = JSON.parse(response1.data);
3 expect(body).to.be.an('array');
4
5 expect(body).to.have.lengthOf.above(1);
6
7 const item = body[0];
8 expect(item).to.be.an('object');
9 expect(Object.keys(item)).to.have.lengthOf(4);
10 expect(item).to.have.property('id');
11 expect(item).to.have.property('username');
12 expect(item).to.have.property('score');
13 expect(item).to.have.property('active');
14 expect(item.username).to.equal('bear');
```

On the right, under 'Tests Failed 1/1', it shows a failed test for 'Fetch All Users' with the error message: 'expected 'bear' to equal 'bearasdf''. The test took 337 ms.

161

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');
```

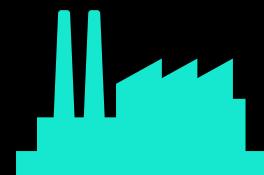
API Design: Automated Testing



- **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**

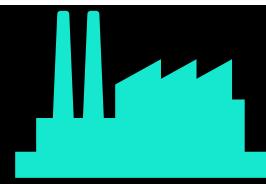
163

API Design: Legacy Systems



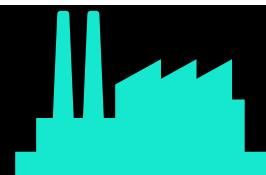
- **What is the Legacy system?**
- **How to map to REST nouns**
 - **What are the data?**
 - **Tables? Columns? Join?**
 - **e.g., Report generation -> /report**
- **Focus on API, state (not functionality)**
- **API might be a facade**
- **Treat the client and server as equals**

164



- **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**

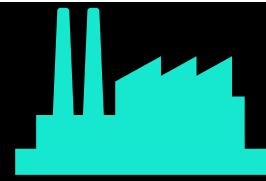
165



- **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**

166

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**

167

API Design: Versioning



168



- **Versioning**
 - **In URL**
 - **Easy, clear**
 - **Requires URL changes for clients**
 - **...example.com/api/v2/reports/34**

169



- **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**

170

API Design: Versioning



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

171

API Design: Versioning



- **Backward Compatible**
 - **Old versions -> old code**
 - **Or new code's version of functionality**
 - **Behind the scenes implementation**

172

API Design: Versioning



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

173

API Design: Security by Design



174

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 ZXh0ZXJuYWwtc2Vydm1jZW5hbWU6YXBpa2V5
```

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

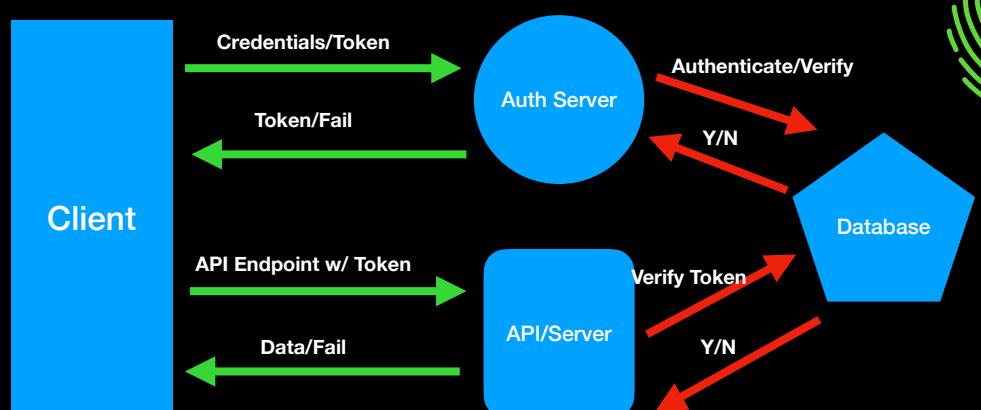
- **JWT**
- <https://jwt.io/introduction>
 - <header>.<payload>.<signature>
- <https://openid.net/developers/jwt/>
- **Authentication/Authorization**
 - **Access - http header**
 - **Expiration - refresh**
 - **SDKs often handle bulk**
 - **Authorize by user/role**



177

API Design: Security by Design

- **Authenticate and Authorize**



178

API Design: Security by Design

- **securityDefinitions** is now **securitySchemes** in components for reuse

```
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
```

179

API Design: Security by Design

- **Example: Cognito**



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

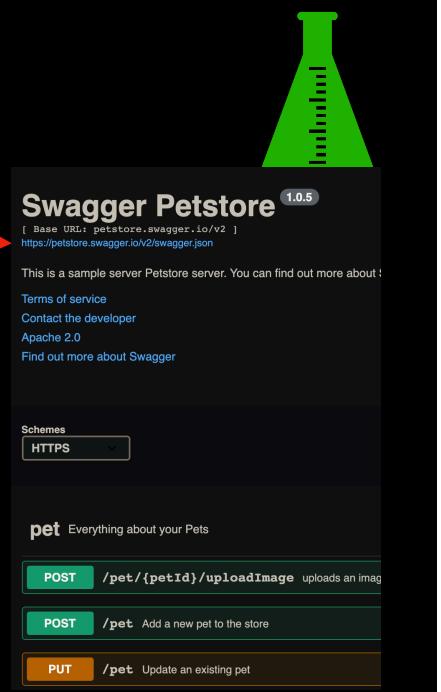
```
paths:  
  /v2/alarm:  
    get:  
      responses:  
        "200":  
          description: "200 response"  
          content:  
            application/json:  
              schema:  
                $ref: "#/components/schemas/Alarm"  
      security:  
        - MyAppAuth: []
```



180

API Design

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



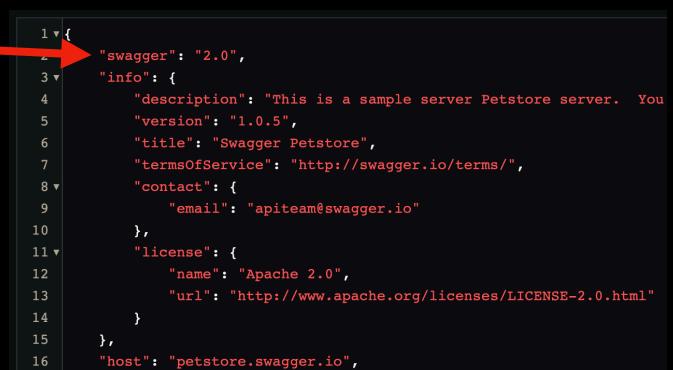
The screenshot shows the Swagger Petstore interface version 1.0.5. At the top, it displays the base URL: petstore.swagger.io/v2 and the JSON file: https://petstore.swagger.io/v2/swagger.json. Below this, there's a navigation bar with links to Terms of service, Contact the developer, Apache 2.0, and Find out more about Swagger. A dropdown menu for Schemes is set to HTTPS. The main content area is titled 'pet' with the subtitle 'Everything about your Pets'. It lists three operations:

- POST /pet/{petId}/uploadImage - uploads an image
- POST /pet - Add a new pet to the store
- PUT /pet - Update an existing pet

181

API Design

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



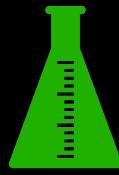
A screenshot of a code editor displaying a Swagger 2.0 JSON file. The JSON structure includes:

```
1 v{  
2   "swagger": "2.0",  
3   "info": {  
4     "description": "This is a sample server Petstore server. You can find out more about APIs",  
5     "version": "1.0.5",  
6     "title": "Swagger Petstore",  
7     "termsOfService": "http://swagger.io/terms/",  
8     "contact": {  
9       "email": "apiteam@swagger.io"  
10      },  
11      "license": {  
12        "name": "Apache 2.0",  
13        "url": "http://www.apache.org/licenses/LICENSE-2.0.html"  
14      }  
15    },  
16    "host": "petstore.swagger.io",  
17  }
```

A red arrow points to the line "swagger": "2.0".

182

API Design



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

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

pet

Swagger Editor documentation

Find out more: <http://swagger.io>

| Method | Path | Description |
|--|--------------------------|---|
| POST | /pet | Add a new pet to the store |
| PUT | /pet | Update an existing pet |
| GET | /pet/findByStatus | Finds Pets by status |
| DELETE | /pet/{petId}/tags | Deletes multiple values from the pet's tags at once |
| GET | /pet/{petId} | Find pet by ID |
| POST | /pet/{petId} | Updates a pet in the store with form data |
| DELETE | /pet/{petId} | Deletes a pet |
| POST | /pet/{petId}/uploadImage | uploads an image |
| store Access to Petstore orders | | |
| GET | /store/inventory | Returns pet inventories by status |
| POST | /store/order | Place an order for a pet |
| GET | /store/order/{orderId} | Find purchase order by ID |
| DELETE | /store/order/{orderId} | Delete purchase order by ID |

183

API Automated Testing



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

184



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



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



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

187



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

188



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



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



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



191



- **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)

192



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

193



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

194

API Automated Testing



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

195

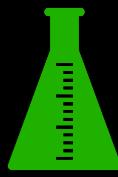
API Automated Testing



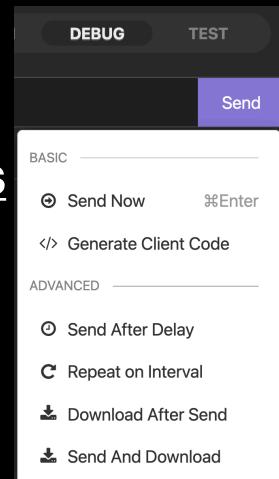
- **Integration Testing**
 - **End-to-End**
 - **Doesn't have to be last**
 - **May take more time than expected**
 - **Maybe (automated) UI driven**
 - **SDK**
 - **Client Agnostic**

196

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>



197

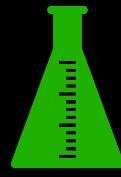
API Design: Automated Testing



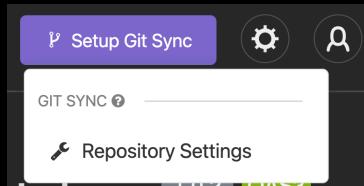
- **Automated Testing with Insomnia**
- **inso CLI**
 - **Lint your spec**
 - **Look for and trap small issues with big effects in syntax and similar**
 - **Run Tests**
 - **CI/CD on checkin, automated**
 - **Define config**
 - **Generate Code, Documentation**

198

API Automated Testing



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



Configure Repository [?](#) [x](#)

Git URI (https)

Author Name
Name

Author Email
Email

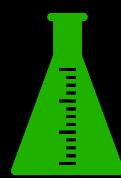
Username
MyUser

Authentication Token [?](#)

Done

199

API Automated Testing



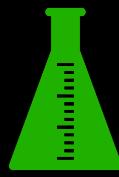
- **Token**
 - **Click ? For links to instructions**
 - **Sync with Github or similar**

Create a personal access token
[Github](#) [Gitlab](#) [Bitbucket](#) [Bitbucket Server](#)

Authentication Token [?](#)

200

API Automated Testing



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

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

```
Bear-MBP:~ bearc2020$ npm install --global insomnia-inso
npm WARN deprecated har-validator@5.1.5: This library is no longer supported
npm WARN deprecated uid-safe@1.0.1: Deprecated due to CVE-2022-21364 resolved in v6.5.0
npm WARN deprecated request@2.88.2: request has been deprecated. This version will be removed in 2024. New versions may use Math.random() in certain circumstances, which is known to be problematic.
See https://v8.dev/blog/math-random for details.
npm WARN deprecated request-promise@0.0.2: request has been deprecated, see https://github.com/request/request#issue/3142
npm WARN deprecated request-promise-native@1.0.1: This package has been deprecated and no longer supports promises now, please switch to that.
npm WARN deprecated request@2.88.1: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.88.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.87.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.86.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.85.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.84.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.83.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.82.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.81.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated request@2.80.0: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated node-pre-gyp@0.15.4: Please upgrade to @mapbox/node-pre-gyp: the non-scoped node-pre-gyp package is deprecated and only the @mapbox scoped package will receive updates in the future.
npm WARN deprecated cryptiles@4.2.1: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated Spectral@0.95.0: Spectral's latest version is now available as @stoplight/spectral-cli on npm
npm WARN deprecated Botlight@spectral@0.95.0: Spectral's latest version is now available as @stoplight/spectral-cli on npm

added 684 packages, and audited 688 packages in 1ss

34 packages are looking for funding
  run npm fund for details

22 vulnerabilities (2 moderate, 9 high, 3 critical)

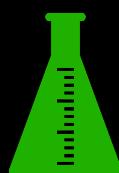
To address issues that do not require attention, run:
  npm audit fix --onlyBrokenDependencies

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
Bear-MBP:~ bearc2020$ npm audit
npm audit report
npm audit audit This command requires an existing lockfile.
```

201

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
```

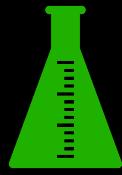
```
Bear-MBP:~ bearc2020$ inso run test
✓ Select a document or unit test suite · Test 1 - uts_276215
✓ Select an environment · OpenAPI env - env_env_65edc1

Test 1
  ✓ Returns 200 (1278ms)

  1 passing (1s)
```

202

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)
```

203

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!
```

204

API Automated Testing



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

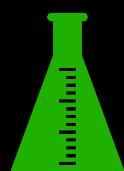
```
scripts:  
  _ mylint: inso lint spec "My API"
```

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

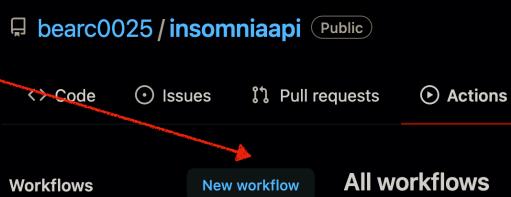
```
|Bear-MBP:insomnia bearc2020$ inso mylint  
No linting errors. Yay!  
|Bear-MBP:insomnia bearc2020$ inso script mylint  
No linting errors. Yay!
```

205

API Automated Testing

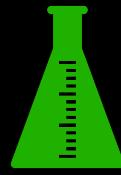


- **Insomnia CI/CD Tests**
 - **e.g., Github Actions**
 - **<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>**
 - **"New Workflow"**



206

API Automated Testing



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

Choose a workflow template

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging

Skip this and [set up a workflow yourself](#) →

Workflows made for your repository Suggested

Simple workflow
By GitHub

Start with a file with the **minimum necessary structure**.

[Set up this workflow](#)

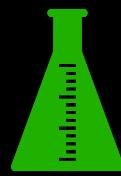
```
echo Hello, world!
echo Add other actions to build,
echo test, and deploy your project.
```

[actions/starter-workflows](#)

```
1 # This is a basic workflow to help you get started with Actions
2
3 name: CI
4
5 # Controls when the workflow will run
6 on:
7   # Triggers the workflow on push or pull request events but only for the master branch
8   push:
9     branches: [ master ]
10    pull_request:
11      branches: [ master ]
12
13 # Allows you to run this workflow manually from the Actions tab
14 workflow_dispatch:
15
16 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
17 jobs:
18   # This workflow contains a single job called "build"
19   build:
20     # The type of runner that the job will run on
21     runs-on: ubuntu-latest
22
23   # Steps represent a sequence of tasks that will be executed as part of the job
24   steps:
25     # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
26     - uses: actions/checkout@v2
27
28     # Runs a single command using the runners shell
29     - name: Run a one-line script
30       run: echo Hello, world!
31
32     # Runs a set of commands using the runners shell
33     - name: Run a multi-line script
34       run: |
35         echo Add other actions to build,
36         echo test, and deploy your project.
```

207

API Automated Testing



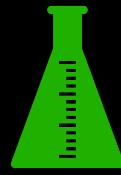
- **<https://docs.insomnia.rest/inso-cli/continuous-integration>**
- **Copy 'steps' from Insomnia example at site above**
- **Replace command parameters with your project names/ids**

```
# .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
```

208

API Automated Testing

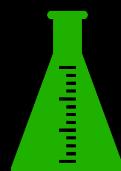


- **Paste steps into template**
- **Optionally remove pull request run in 'on' section**
- **On push...**
 - **Checkout**
 - **Install**
 - **Lint**
 - **Tests**
 - **Generate config**

```
1 # This is a basic workflow to help you get started with Actions
2
3 name: CI
4
5 # Controls when the workflow will run
6 on:
7   # Triggers the workflow on push or pull request events but only for the master branch
8   push:
9     branches: [ master ]
10
11 # Allows you to run this workflow manually from the Actions tab
12 workflow_dispatch:
13
14 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
15 jobs:
16   # This workflow contains a single job called "build"
17   build:
18     # The type of runner that the job will run on
19     runs-on: ubuntu-latest
20
21 # Steps represent a sequence of tasks that will be executed as part of the job
22 steps:
23   - name: Checkout branch
24     uses: actions/checkout@v1
25   - uses: kong/setup-inso@v1
26     with:
27       inso-version: 2.4.0
28   - name: Lint
29     run: inso lint spec "My API" --ci
30   - name: Run test suites
31     run: inso run test uts_276215 --env "OpenAPI env" --ci
32   - name: Generate declarative config
33     run: inso generate config "My API" --type declarative --ci
```

209

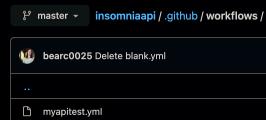
API Automated Testing



- **Change the name of the file**

insomniaapi / .github / workflows / **blank.yml** in **master**

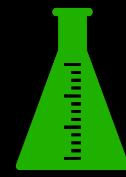
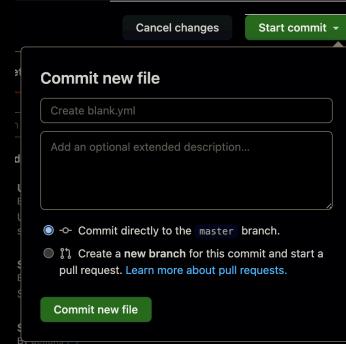
- **When saved, .github/workflows with that file will be added to your repo**



210

API Automated Testing

- **Commit the file**
- **Add name and comments as you like**
- **'Commit new file'**
- **Pull in Insomnia to keep in sync!**



211

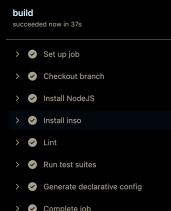
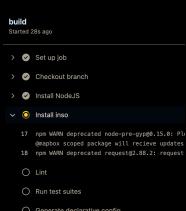
API Automated Testing

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

🟡 new github action

CI #7: Commit 9c97b1b pushed by bearc0025

- **Click it for details**



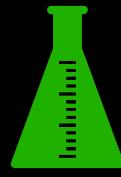
- **Actions shows status**

🟢 new github action

CI #7: Commit 9c97b1b pushed by bearc0025

212

API Automated Testing



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

- **test update for action**
CI #8: Commit 15e7abf pushed by bearc0025

- **Verify success (or failure)**

- **test update for action**
CI #8: Commit 15e7abf pushed by bearc0025

- **End of lab**

A screenshot of a GitHub CI workflow run. At the top, it shows a GitHub icon and the text "[bears0025/insomniaapi] CI workflow run". Below that, there's a large green button with a play icon. To the right, it says "CI: All jobs have failed" and has a "View workflow run" button. At the bottom, there's a red circle with an "X" and the text "CI / build Failed in 14 seconds" followed by a small icon with the number "1".

213

Tutorials

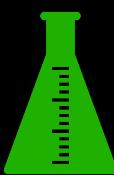
Jetty Web API Servlet

jetty://

API Design: Java Server and Client

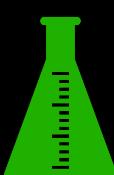
- **Lab: Java Server**
- **Note: This is not meant to be an exhaustive step-by-step for configuring Jetty but may be helpful:**
 - Jetty: <https://www.eclipse.org/jetty/>
 - Unzip
 - Set env var:
 - JETTY_HOME=/Users/bearc2020/Downloads/jetty-home-11.0.5
 - Start server:
 - java -jar start.jar
 - Load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





- **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`

217



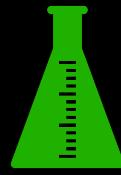
- **Under the jetty-base/webapps dir**
 - **Create a PersonServlet dir**
 - **Inside PersonServlet create src and WEB-INF dirs**
 - **src is for source code but can be located anywhere**
 - **WEB-INF is for configuration, classes and libraries**



218

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**

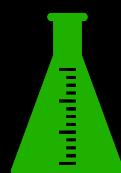


```
AuthServlet.class  
AuthServlet.java  
DataStore.class  
DataStore.java  
java-json.jar  
Person.class  
Person.java  
PersonServlet.class  
PersonServlet.java  
RestApiClient.class  
RestApiClient.java  
TokenStore.class  
TokenStore.java  
TokenStore...nData.class
```

219

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 servlet api and json jar files.**
- **Copy the class files into**
 - **WEB-INF/classes**

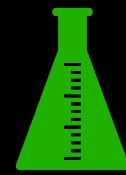


```
AuthServlet.class  
DataStore.class  
Person.class  
PersonServlet.class  
RestApiClient.class  
TokenStore.class
```

220

API Design: Java Server and Client

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

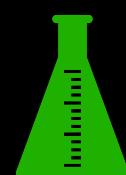


```
<web-app
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
```

221

API Design: Java Server and Client

- **Servlet(s)...**



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

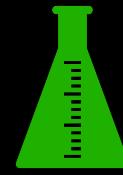
<servlet-mapping>
    <servlet-name>PersonServlet</servlet-name>
    <url-pattern>/people/*</url-pattern>
</servlet-mapping>

<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>
```

222

- **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>
```

223

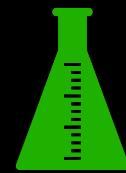
- **With the env var, you can start jetty from other dirs.**
- **Restart jetty from the jetty-base dir:**
 - **java -jar \$JETTY_HOME/start.jar**
 - **Load: http://localhost:8080/**
- **Load the web app at:**
 - **http://localhost:8080/PersonServlet/people/Kevin**



224

API Design: Java Server and Client

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



API Client

Token:

Name:

Password:

225

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, "pkw"));
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**

226

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>**

Visual Studio Visual Studio Code Visual Studio for Mac

- Visual Studio Code ↗
- C# for Visual Studio Code (latest version) ↗
- .NET 5.0 SDK ↗

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.helorestjava -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!
```

231

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>
```

232

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)**

233

Java Standalone Server



- **Spark**
- **Error**

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-compile) 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>
```

234

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**

235

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: 450ba27947e13e66baa8cd1ce7e85a4461caccid; 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

236

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}
```

237

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}
```

238



- **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

239



- **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**

240

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"));
});
```

241

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>**

242

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 flask 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)
```

245

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)
```

246

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
```

247

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"
  }
]
```

248

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>')
```

249

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"
}
```

250

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"
}
```

253

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)
```

254

RESTful API Design and Development

The End

