Welcome
Introduction to API Design and Development

DevelopIntelligence
A PLURALSIGHT COMPANY

You?

- Name
- Total experience
- Background in API design & development
- Goals

Hello...

About me…

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

Prerequisites

This course assumes you:

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

Why RESTful JSON?

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

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

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

Objectives

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

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

Agenda

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

How we're going to work together

- Discussion
- Demo
- Do!

Breaks

10 mins break every hour

Housekeeping

📞 If I drop, I'll be back

Working with virtual machines

We will be doing most of our work in virtual machines

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

DevelopIntelligence

A PLURALSIGHT COMPANY

Introduction to API Design and Development

Understanding APIs in Software Delivery: Introduction

Understanding APIs in Software Delivery: Introduction

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

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

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

API Design: Introduction

API Design: Introduction

API Design: Introduction

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

API Design: Introduction

- **API Routing & Security**

DNS

Load Balancing

Scalability

Authentication

OAuth, Token/JWT

Authorization

Permissions

API Design: Introduction

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

API Design: Introduction

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

API Design: Introduction

API Design: Introduction

API

OpenAPI Spec

API Design: Introduction

- **AWS**

API Design: Introduction

- **Past:**

  API Wild West

  URL/Query String, Form Data, etc.

  SOAP/WSDL

  XML, text, etc.

Everything was proprietary, complex

Needed a Standard

API Design: Introduction

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

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

API Design: Introduction

Early 2000s - Money to be made

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

eBay, Flickr, Amazon, Twitter, etc.

- **The standard is key**

API Design: Introduction

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

Libraries/frameworks/SDKs abound

Variations cause problems

Downside: better solutions will suffer from inertia

API Design: Introduction

gRPC 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.

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

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

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

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

API Design: Introduction

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

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

API Design: Introduction

- **Editor: Info**

editor.swagger.io

Start with version

Errors

Insert > Info

Swagger Editor
Supported by SMARTBEAR

```
1  openapi: 3.0.1
```

Errors

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

API Design: Introduction

- **Editor**

Insert > Info

Enter text in fields

License/Contact optior

Click 'Add Info'

- **Text inserted**

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

End of lab

API Design: JSON

- RESTful JSON

JSON

JavaScript Object Notation

json.org

Arrays, Dictionaries of values

Represented in text

Parseable across platforms, languages, etc.

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

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

members
    member
    member ',' members

member
    ws string ws ':' element

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

API Design: JSON

- **JSON**

Object Example:

User

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

API Design: JSON

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

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

API Design: JSON

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

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

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

members
    member
    member ',' members

member
    ws string ws ':' element

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

API Design: JSON

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

API Design: JSON

- **Lab**

  https://jsonlint.com/

  Create an array of Elements

  Each element has 3 members:

  name of type string

  description of type string

  age of type integer

  See next slide for hints

API Design: JSON

- **Lab**

  json.org has details

  Arrays are within [and]

  Elements are within { and }

  Element members are key-value pairs

  "name" : "test"

  Members and Array elements are comma separated

  See next slide for example

API Design: JSON

- **Lab**

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

End of lab

API Design: Design

- **Key Concepts**

HTTP for communication

State is…

Value(s) of data at a given time

Cookies, Parameters, etc. represent state

REST Statelessness

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

API Design: Design

Key Aspects

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

API Design: Design

Key Aspects

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

API Design: Design

Key Aspects

- Example: DB Tables
  - .../customers
    - .../customers/{customer_id}
  - .../staffs
  - .../orders
  - .../stores
  - .../orderItems
- Could be other resources

API Design: Design

- **Key Aspects**

Nouns - resource based

Relationships in URL

Comments relates to Post 1

/posts/1/comments - returns 2 comments

/comments/1/posts - returns 1 post

Data Source ->

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

API Design: Design

Verbs - CRUD

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

Data Source ->

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

API Design: Design

Verbs - CRUD

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

Data Source ->


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

Lab: GET

API Design: Design

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

View db.json file

Via browser:

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


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


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

No registration. Nothing to install.

Sponsor

Share

Fork

demo

by typicode

Available resources

- posts 3

API Design: Design

Verbs - Create, Update

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


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

API Design: Design

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

POST/PUT/PATCH
- **Overlap**
- **Require HTTP body**


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

API Design: Design

Query String

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

.../api/posts/1?_embed=comments

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

API Design: Design

- All defined in…

OpenAPI Spec

Contract Description

Information

Servers

Paths & Methods

Model

Security

All Interested Parties

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

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

- **Now that we have one…**

Add Server

editor.swagger.io

Insert > Add Servers

Add:

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

API Design: Design Concepts

- **Server**

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

End of Lab

API Design: Design Concepts

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

Creates:

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

API Design: Design Concepts

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

Creates:

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

API Design: Design Concepts

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

- **End of lab**

For more… PluralSight

APIs- Application Programming Interface: Executive Briefing

Dan Appleman

API Design: Design Concepts

API Design: Design Concepts

Separation of Concerns

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

API Design: Design Concepts

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

API Design: Design Concepts

Separation of Concerns

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

API Design: Design Concepts

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

AWS - Separation of Concerns

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

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

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

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

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

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

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

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

Errors

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

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

Errors - Best Practices

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

Errors - Best Practices

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

API Design: Design Concepts

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

API Design: Design

Demo:

GitHub Repo with db.json file

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

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

API Design: Design

Demo:

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

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

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

API Design: Design

Lab

If you have a git repo…

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

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

How to

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

API Design: Design Concepts

- **Now that we have two…**

Add Another Server

editor.swagger.io

Insert > Add Servers

Add your server or mine:

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

- **Server**

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

End of Lab

API Design: Richardson Maturity Model

API Design: Richardson Maturity Model

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

Four Levels:

Level 0: Uses HTTP

Level 1: Resources (nouns)

Level 2: HTTP Verbs

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

API Design: Richardson Maturity Model

- **Level 0: Uses HTTP**

Mostly just HTTP plus anything…

XML, YAML, JSON, etc.

RPC

Might be one endpoint for all requests

Content specifies details for request and response

And/or GET query string

API Design: Richardson Maturity Model

- **Level 1: Resources (nouns)**

Various endpoints

e.g., /posts, /posts/1

Body/GET still specifies details

Request, Response

Various formats: XML, etc.

API Design: Richardson Maturity Model

- **Level 2: HTTP Verbs**

GET

Same request, same results

Cacheable

POST (create)

201 Created

409 Conflict

PUT, PATCH, DELETE, etc.

API Design: Richardson Maturity Model

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

What to do next and how

Can be dynamic

Client doesn’t need knowledge

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

API Design: Richardson Maturity Model

- **HATEOAS**

  No official OpenAPI format

  Common practice includes:

  href: URI

  rel: relation type

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

  See RFC5988 and Hypertext Application Language (HAL)

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

API Design: Richardson Maturity Model

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

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

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

API Design: Richardson Maturity Model

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

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

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

API Design: Richardson Maturity Model

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

Links can be reused

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

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

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

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

API Design: Design

Lab

https://pokeapi.co/

Fetch data and drill down into more

Try with limit and offset

API Design: Design

- Lab

https://swapi.dev/

Fetch data and drill down into more

Try with page query string parameter

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

API Design: Dev Standards

API Design: Dev Standards

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

API Design: Dev Standards

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

API Design: Dev Standards

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

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

API Design: Dev Standards

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

API Design: Dev Standards

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

API Design: Dev Standards

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

API Design: Dev Standards

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

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

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

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

9.2 OPTIONS

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

Responses to this method are not cacheable.

API Design: Dev Standards

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

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

API Design: Dev Standards

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

API Design: Contracts

Describes/Defines the API

- OpenAPI 3.0 (Swagger tools)
- JSON, YAML and others
- Includes info, server, paths, etc.

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

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

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

API Design: Contracts

OpenAPI Documentation

https://swagger.io/specification/

Various sections and types

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

API Design: Contracts

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

API Design: Code First

Create API in Insomnia

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

API Design: Code First

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

API Design: Code First

- **Test on the right**

End of lab

API Design: Contracts

- **For more… PluralSight**

Designing RESTful Web APIs

Shawn Wildermuth

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

API Design: RESTful API Basics

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

Getting started

Install JSON Server

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

Create a `db.json` file with some data

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

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

API Design: RESTful API Basics

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

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

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

  \{^_^}/ hi!

  Loading db.json
  Done

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

  Home
  http://localhost:3000

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

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

API Design: RESTful API Basics

- **Lab**

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

- **Lab**

Add Server to Swagger editor

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

API Design: RESTful API Basics

- **Lab**

Test against local server

End of lab

API Design: RESTful API Basics

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

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

API Design: RESTful API Basics

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

Request Body Object

Describes a single request body.

Fixed Fields

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

API Design: RESTful API Basics

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

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

API Design: RESTful API Basics

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

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

API Design: RESTful API Basics

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

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

API Design: RESTful API Basics

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

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

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

API Design: RESTful API Basics

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

- **End of lab**

API Design: RESTful API Basics

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

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

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

- **End of lab.**

API Design

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

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

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

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

API Design

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

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

API Design

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

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

API Design

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

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

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

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

- **End of lab**

API Design

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

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

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

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

- **End of lab**

API Design

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

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

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

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

- **End of Lab**

API Design

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