Designing APIs with OpenAPI

Mike Amundsen
@mamund


Mike Amundsen
@mamund

OPENAPI INITIATIVE

About Specifications Participate Governance Membership Blog FAQ What is OpenAPI? Events

Want to learn more about OpenAPI? Take our new OpenAPI Fundamentals training course LEARN MORE >

The world's most widely used API description standard

View Latest OpenAPI Spec | How to Get Involved | View Latest Arazzo Spec | View New Overlays Spec

What is OpenAPI?

The OpenAPI Specificati
a formal standard for de

apidays

The World's Leading API Conference Series

FEATURED IN:

Forbes VB TechCrunch

Next Conference

APIDAYS PARIS - DECEMBER 3 - 5, 2024

The Future API Stack for Mass Innovation

APPLY TO BE A SPEAKER | APIDAYS 2024 EVENTS | SPONSORSHIP DECK

15 countries

apimasters.

Courses Paris 2024 Workshops Conferences Replays Login/Register

apimasters. by apidays

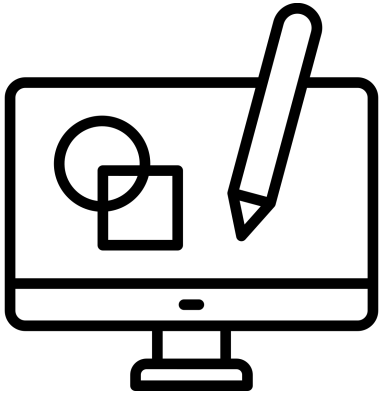
Skill Up for the Future

Learn and master APIs to reach the next level

Learn new skills

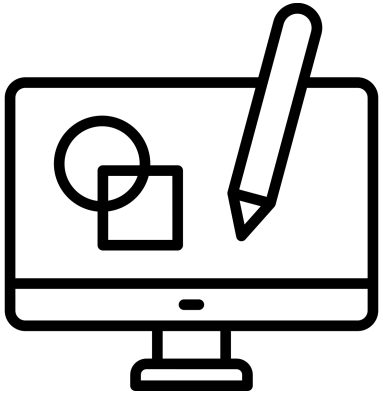
Designing APIs with OpenAPI

- The Power of API Design
- API Design Assets
- Prototyping Your API Design with OpenAPI
- Leveraging OpenAPI Components
- Defining OpenAPI Paths
- Mocking and Testing Your API Design
- Modifying Your API in Production
- Review



The Power of API Design

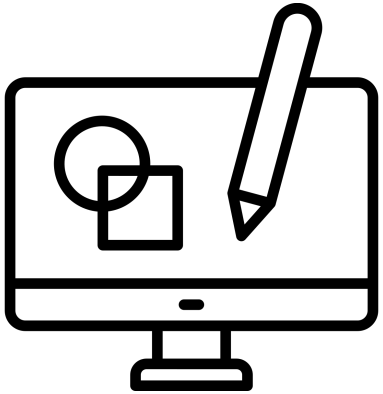
The Power of API Design



- API-First
- Norman's Lifecycle
- Three Pillars of API Design
- Exercise: Identify Your Three Pillars

API First

- Identify key actors
- Before you build …
- First class citizens



blogorrhea: API-First Design

Not secure | asserttrue.blogspot.com/2009/04/api-first-design....

Wednesday, April 01, 2009

API-First Design

I don't pretend to be an architect, but sometimes I wonder if APIs shouldn't be the starting point of product development, rather than an afterthought.

Yesterday, I suggested that if there were such a thing as a middleware API maturity model, "intentional API design" would have to be one of the middle levels. But I think (in fact I know) it's possible to do better than that. It's possible to make API design the sharp edge of the wedge, in software development.

This is not a new concept. Joshua Bloch and others have advocated *early* design of APIs, rather than letting APIs trail product design (as currently practiced). By early design, I mean that the APIs literally are written first, before the software itself.

A lot of software development happens this way, of course: You draw UML diagrams, identify key interfaces, write mock methods, etc., before writing "real" code. That's fine, but it's not quite the same as defining an API.

API-first design means identifying and/or defining key actors and personas, determining what those actors and personas expect to be able to *do* with APIs (i.e., what are the possible use-cases, user narratives, or stories that encapsulate the business problems these people need to solve), and -- very important -- trying to understand the mental model each actor brings to the problem space. The mental model will drive architectural and design decisions at various levels (for example, it will suggest what kinds of *business objects* each user thinks in terms of) and will keep the overall process on a track toward good usability.

http://asserttrue.blogspot.com/2009/04/api-first-design.html

"API-first design means identifying and/or defining key actors and personas, determining what those actors and personas expect to be able to do with APIs"

-- Kas Thomas, 2009

http://asserttrue.blogspot.com/2009/04/api-first-design.html

What Is An API First Strategy? A

apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-...

API EVANGELIST

What Is An API First Strategy? Adding Some Dimensions To This New Question

I hear this phrase used more and more lately, "An API First Strategy". It is something you hear from startups like printing API LOB, to established players like Pearson, all the way up to 18F at the GSA in Washington DC, but what is API first? Much like APIs in general, an API first strategy is something that will mean something different to each company, organization, government agency, or individual.

{A}1ST

From my view, the most common interpretation of API first will be:

- **Develop an API First** - Before you build your website, web, mobile or single page application you develop an API first, then you get to work on defining the channels you will be making the API resources available on.

I had an interesting conversation today with the 18F team in Washington DC, and they illuminated some other possible dimensions to the API first conversation for me, that I wanted to explore a little:

- **Design Your API First** - Using tools like Apiary, and API definition formats like API Blueprint, you kick off discussion with customers, generate user stories, and mockup your API, and even your web or mobile app interfaces, using your mock API.
- **API First In Open Source Software -** Encouraging the development and use of open source software

https://apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-to-this-new-question/

"Before you build your website, web, mobile or single page application you develop an API first."

-- Kin Lane, 2014

https://apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-to-this-new-question/

Understanding the API-First Approach to Building Products

By Janet Wagner

Table of Contents

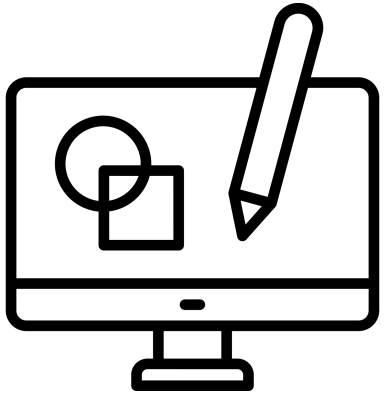
https://swagger.io/resources/articles/adopting-an-api-first-approach/

"An API-first approach means that for any given development project, your APIs are treated as 'first-class citizens.'"

-- Janet Wagner

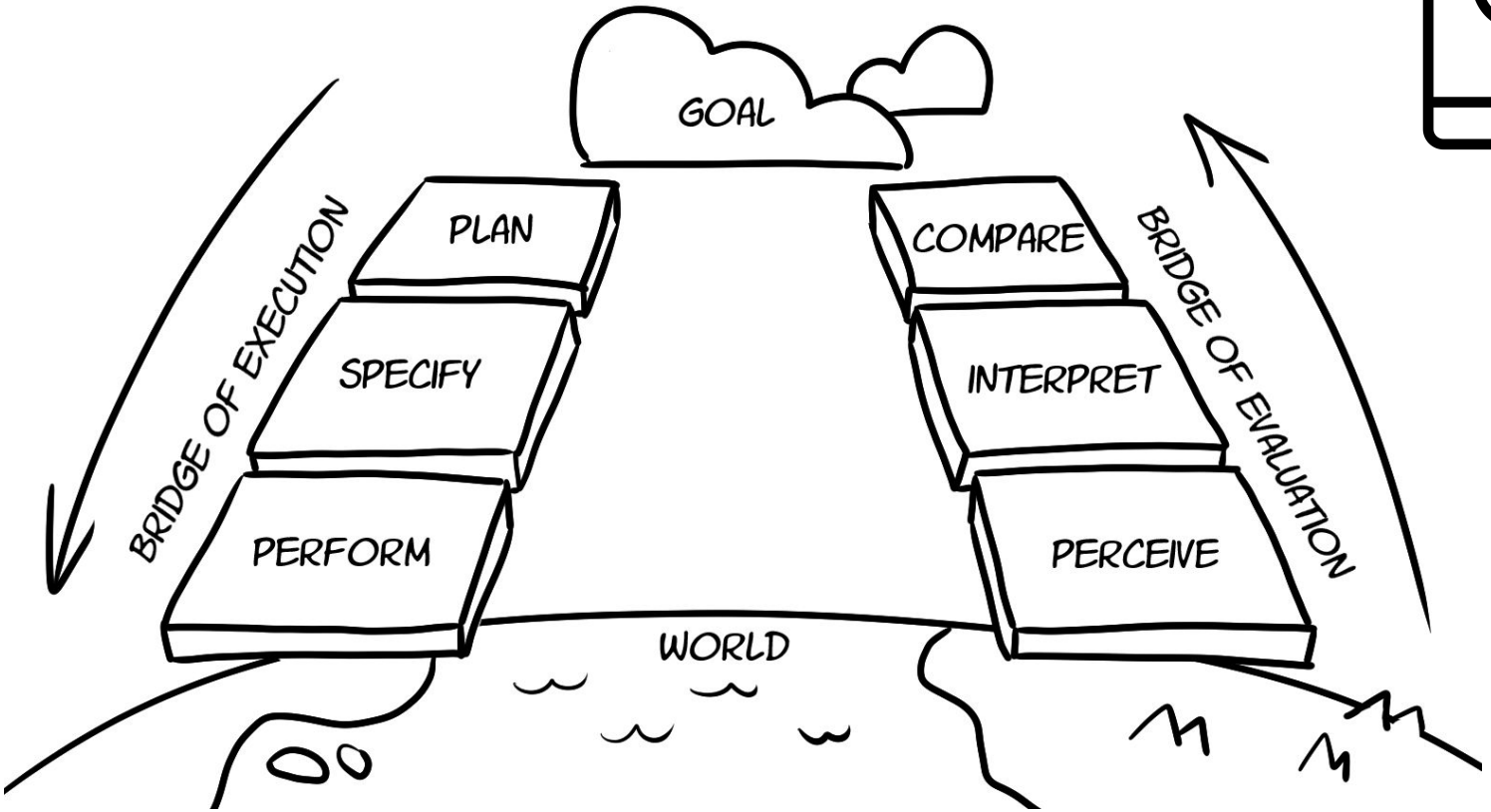
https://swagger.io/resources/articles/adopting-an-api-first-approach/

Norman's Lifecycle

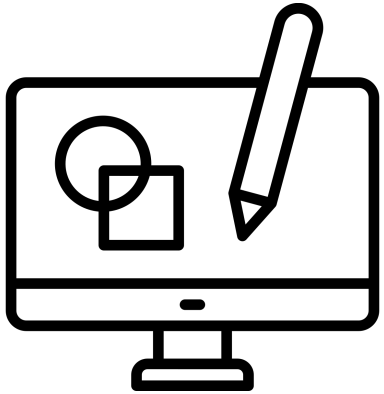


- Donald Norman's Action Lifecycle
- Three stages
  - Goal formation
  - Execution
  - Evaluation

Norman's Lifecycle



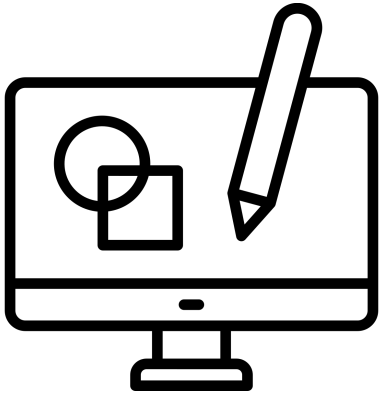
Norman's Lifecycle



- Donald Norman's Action Lifecycle
- Three stages
  - Goal formation
  - Execution
  - Evaluation

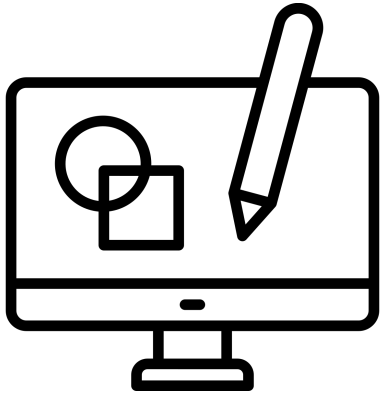
Identify action lifecycles

Three Pillars of API First



- Business Problem
  - What business problem does this API solve?
- Target Audience
  - Who will use this API? And why?
- Actual Design
  - What designs solve this problem for this audience?

Exercise: Identify Your Three Pillars

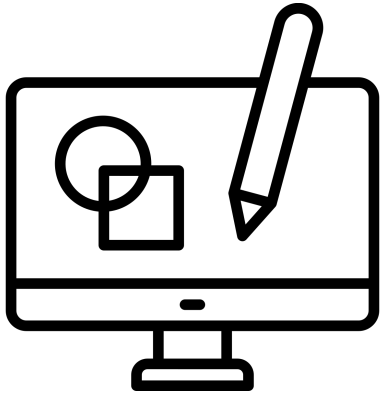


Create a short document that :

1. Identifies a business problem to solve w/ your API
2. Defines a target audience for your API
3. Describes some key features of your API solution

No more than ~150 words

The Power of API Design : Review



- API-First
  - First class citizen
- Norman's Lifecycle
  - Everything is a loop
- Three Pillars of API Design
  - Business, audience, design features

API Design Assets

Design Assets

Design assets are any digital resources that are created or used in the design process.

-- Uxcel

https://app.uxcel.com/glossary/assets

API Design Assets

- API Story
- API Diagram
- API Vocabulary
- Exercise: Assemble Your Design Assets

API Story

- APIs start with a story
  - "We need…"
  - "Our customers requested…"
  - "I have an idea…"
- Stories are shared understanding
  - Our brains are wired for stories, not data
  - Stories are accessible
  - Stories are repeatable

API Story

- Purpose
  - Short description of the purpose/goal
- Actions
  - A list of all the supported actions
- Data
  - A list of all the data elements
- Rules
  - A list of rules that govern the data mgmt
- Processing
  - Any additional processing handled by the API

Person Service API

Purpose

The Person Service allows users to keep track of important people.

Action

The Person Service API supports the following actions:

- Create Person
- Get Person List
- Filter Person List
- Remove Person
- Get Person
- Update Person
- Update Person Status
- Remove Person

Data

The service keeps tracks of important persons. For each person there is a unique `id` for each person record stored. Each record also contains values for `givenName`, `familyName`, `telephone`, and `email`. Each record also has an internal `status` value.

Rules

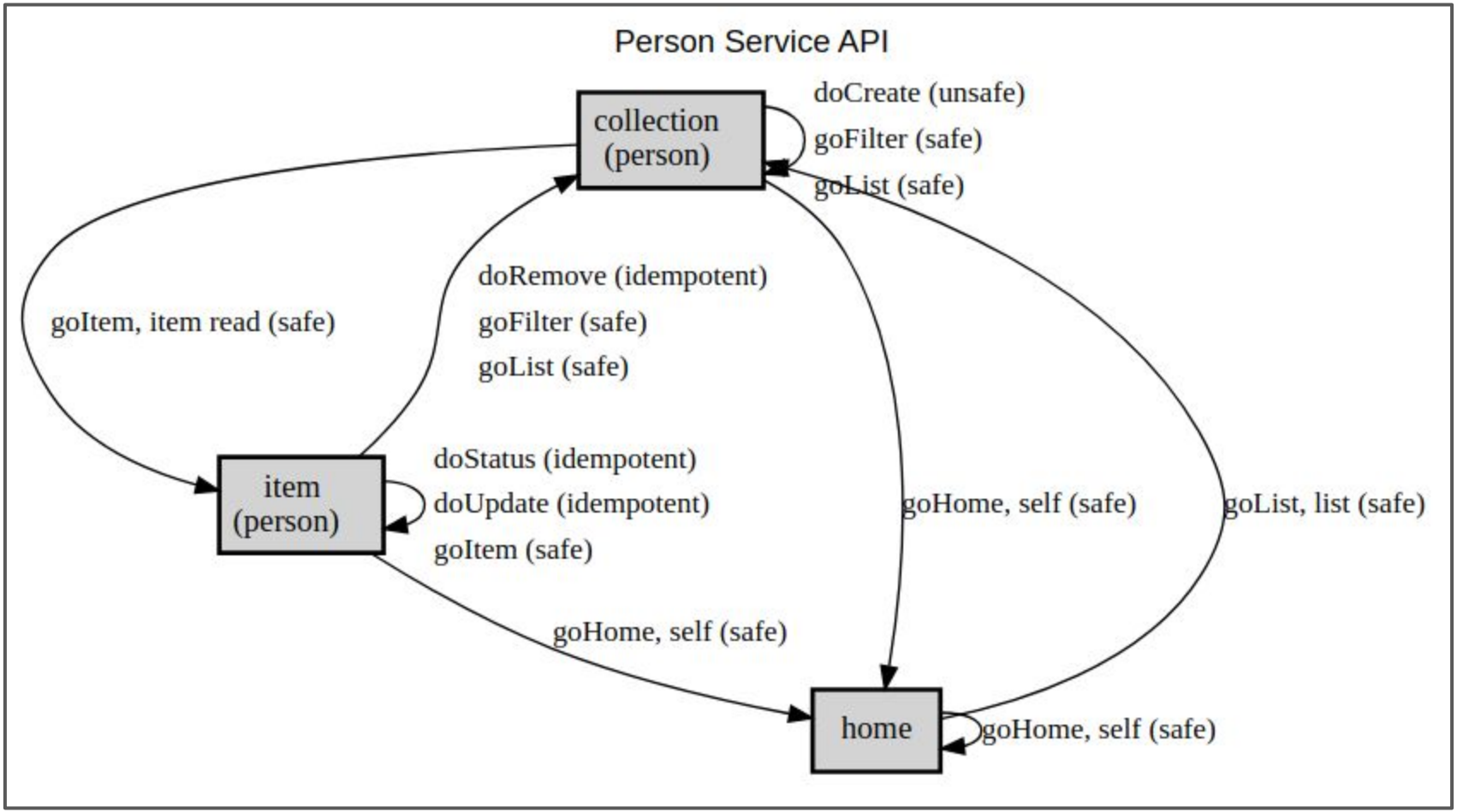
- When a new record is created the client application can establish the value for `id` or allow the service to assign one. If the client attempts to apply an `id` value that is alreay in use, the new service will reject the record.
- The only valid values for `status` are `active` and `inactive`. Any other value will be rejected by the service.
- The following fields are required when storing updated or new records: `id`, `givenName`, `familyName`, `status`

Processing

NONE

API Diagram

- Visual representation of the API
- Not a state diagram
- Not a flow diagram
- An interaction diagram


Person Service API
collection (person)
doCreate (unsafe)
goFilter (safe)
goList (safe)
goItem, item read (safe)
doRemove (idempotent)
goFilter (safe)
goList (safe)
item (person)
doStatus (idempotent)
doUpdate (idempotent)
goItem (safe)
goHome, self (safe)
goHome, self (safe)
goList, list (safe)
home
goHome, self (safe)

API Vocabulary

- All the "magical words" related to API domain
- Not just the properties:
  - id, givenName, etc.
- Also :
  - Actions (create, approve, save, etc.)
  - Enumerators (status.active, status.inactive)
  - Containers (item.id, item.givenName, etc.)
  - Resources/Topics (home, collection. item)

API Person Service Vocabulary

Resources

- home (semantic), Home (starting point) of the person service
- collection (semantic), List of person records
- item (semantic), Single person record

Actions

- doCreate (unsafe), Create a new person record
- doRemove (idempotent), Remove an existing person record
- doStatus (idempotent), Change the status of an existing person record
- doUpdate (idempotent), Update an existing person record
- goFilter (safe), Filter the list of person records
- goHome (safe), Go to the Home resource
- goItem (safe), Go to a single person record
- goList (safe), Go to the list of person records

Containers

- person (semantic), The properties of a person record

Properties

- id (semantic), Id of the person record
- familyName (semantic), The family name of the person
- givenName (semantic), The given name of the person
- email (semantic), Email address associated with the person
- telephone (semantic), Telephone associated with the person
- status (semantic), Status of the person record (active, inactive)

Exercise: Assemble Your API Design Assets

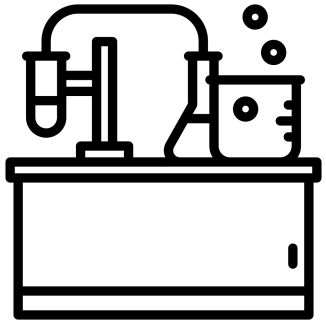
Create an API Story for your API

1. Purpose
2. Actions
3. Properties (Data)
4. Rules
5. Processing

1-3 are required, 4 & 5 are optional

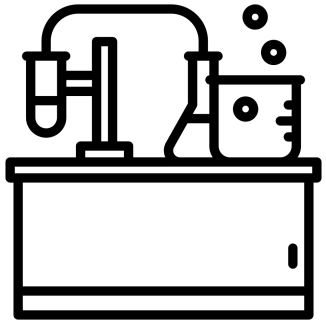
API Design Assets : Review

- API Story
  - Shared, accessible, repeatable
- API Diagram
  - Display the interactions
- API Vocabulary
  - The "magical words" of the API



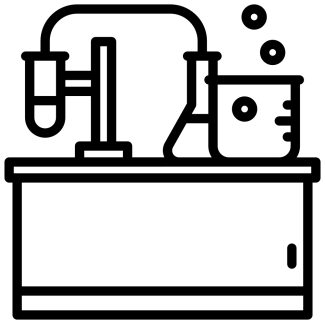
Prototyping Your API Design

Prototyping Your API Design



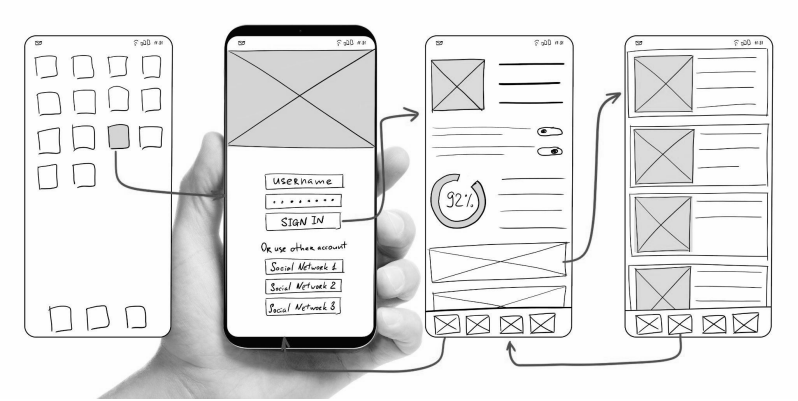
- What is an API Prototype?
- Why Use Prototypes?
- OpenAPI Info Section
- Exercise : Create Your OpenAPI Document

What is an API Prototype?

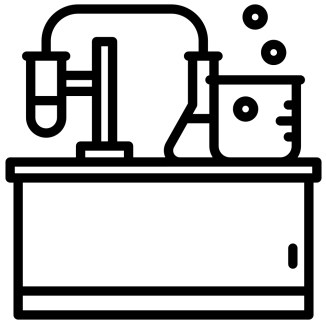


A first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.

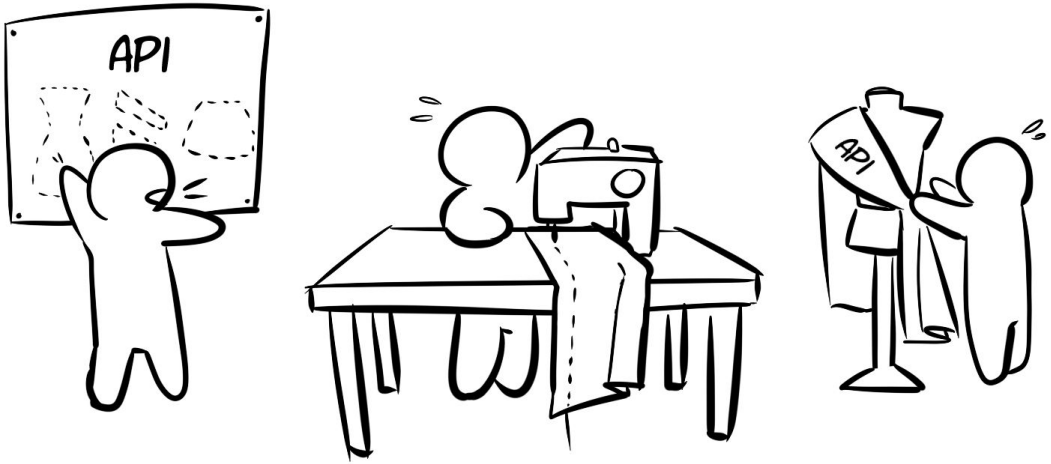
-- Oxford Dictionary



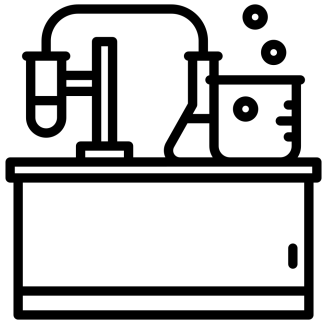
Why use prototypes?



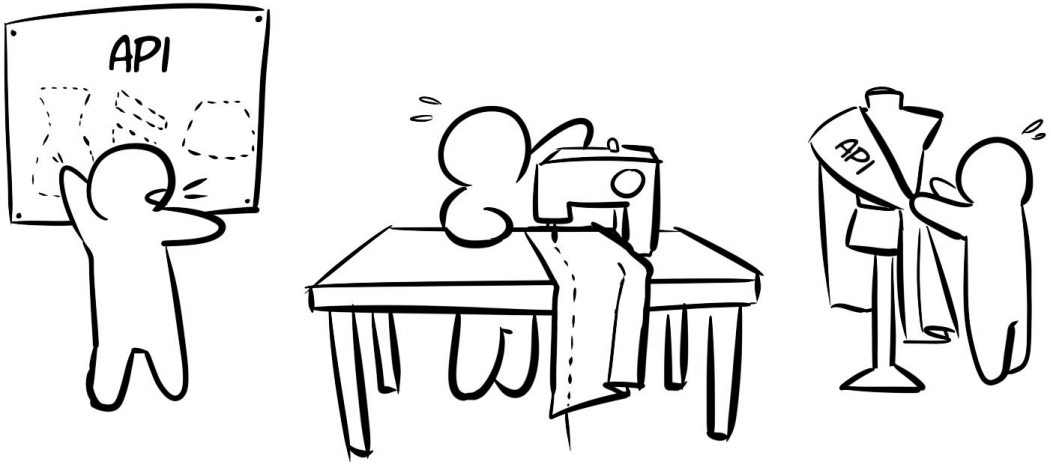
- Inexpensive experiments
- Explore the details
- Prototypes are made to be tested



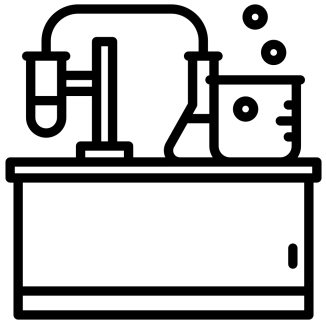
Why use prototypes?



- Inexpensive experiments
- Explore the details
- **Prototypes are made to be tested**



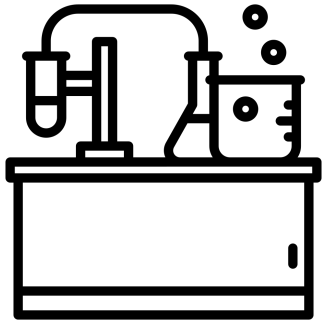
OpenAPI Basic Info



- Contains the basic shared information about your API
- Key elements to include are:
  - `version`
  - `title`
  - `description`
- Other elements:
  - `servers`
  - `paths`
  - `components`

```yaml
openapi: 3.0.0
info:
  version: '1.0'
  title: Person API
  description: An example API for the "Designing APIs with OpenAPI" workshop
servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/amundsen/Person-API/1.0

paths: {}
components: {}
```

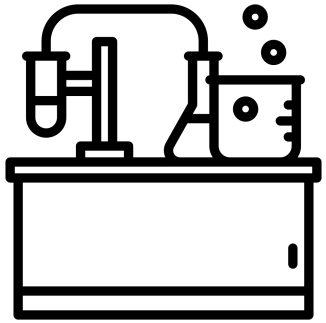
Exercise: Create Your OpenAPI Document



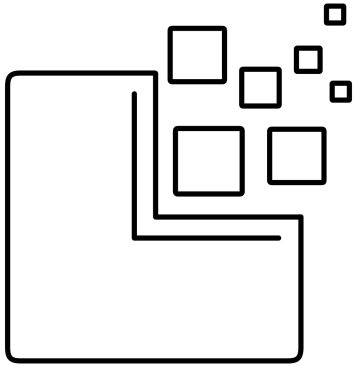
Create the initial OpenAPI document for your API

- Open the SwaggerHub editor
- Log in
- Select "Create New -> Create New API"
- Fill in the dialog
  - See exercise document for details

Prototyping Your API Design : Review

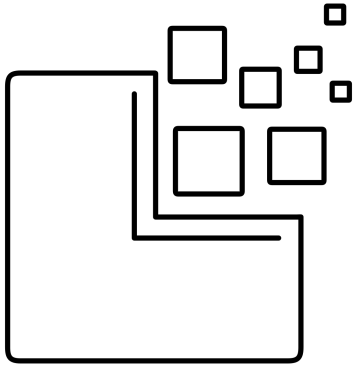


- What is an API Prototype?
  - A preliminary model
- Why Use Prototypes?
  - Prototypes are made to be tested
- OpenAPI Info Section
  - Version, title, description



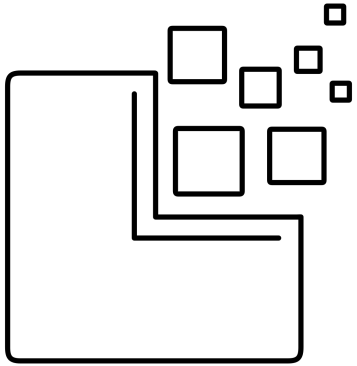
Leveraging OpenAPI Components

Leveraging OpenAPI Components



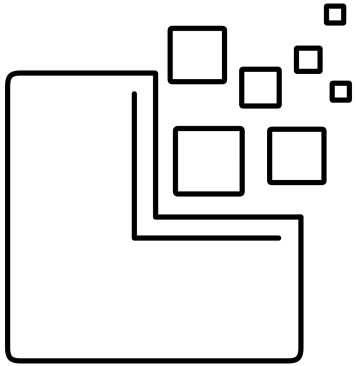
- The OpenAPI Components Section
- Parameters, Schemas, Responses, & RequestBodies
- Exercise: Populate your OpenAPI Component Section

The OpenAPI Component Section



- The `components` section helps organize your API definition
- Establish consistency in your API design
- `parameters` : typically for query strings (?id=123)
- `schemas` : typically for response objects (person, job, etc.)
- `responses` : for other replies (e.g. error messages)
- `requestBodies` : for sending data in POST

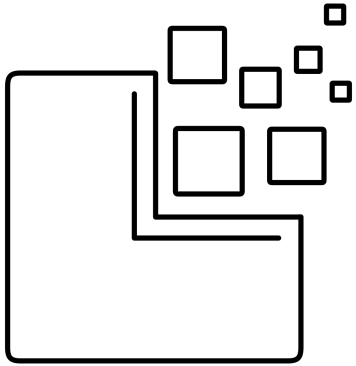
The OpenAPI Component Parameters



- Important properties:
  - name, in, required, schema/type
- Recommend properties:
  - Description
  - example

```yaml
parameters:
  nameParam:
    name: name
    in: query
    description: "Used to filter the list of persons"
    required: false
    schema:
      type: string
      example: "fred"
```

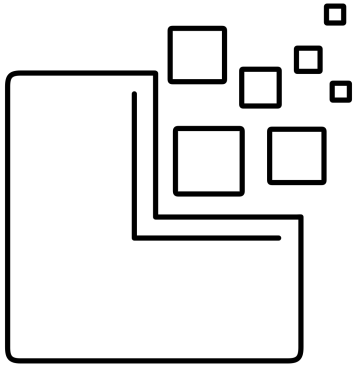
The OpenAPI Component Schemas



- Important properties:
  - Type (object/array)
  - Properties (type, description, example)

```yaml
PersonItem:
  type: object
  properties:
    id:
      type: string
      description: "Unique identifier for each record"
      example: "q1w2e3r4"
    name:
      type: string
      description: "Name of the person"
      example: "Morton Muffley"
    email:
      type: string
      description: "Default email for this person"
      example: "morton.muffley@example.org"
PersonCollection:
  type: array
  items:
    $ref: '#/components/schemas/PersonItem'
```

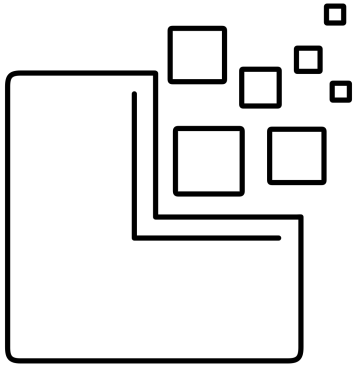
The OpenAPI Component Responses



- Important properties:
  - Content (media type)
  - Schema (use $ref)
- Also:
  - Include description

```yaml
responses:
  GenericError:
    description: "generic error occured"
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/Error"
```

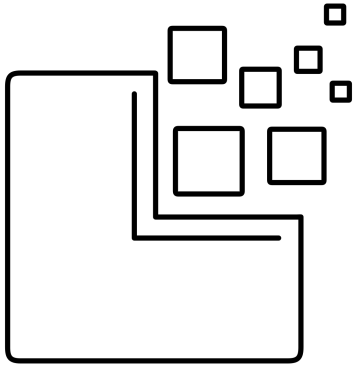
The OpenAPI Component RequestBodies



- Use to support arguments in POST/PUT requests
- Important properties:
  - Description, required, content, schema
- Note: you can support more than one content element

```yaml
requestBodies:
  AddPerson:
    description: "body for a new Person record"
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/PersonItem"
      application/x-www-urlencoded:
        schema:
          $ref: "#/components/schemas/PersonItem"
```

Exercise: Populate OpenAPI Components

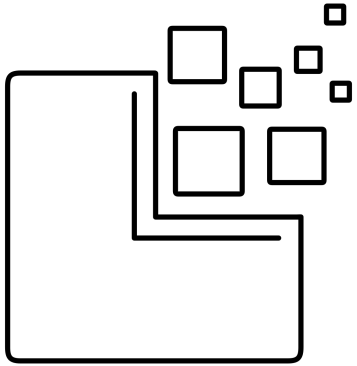


Update your OpenAPI document component section with:

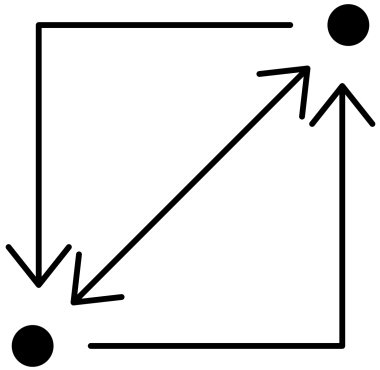
- At least one query parameter (e.g. ?id=)
- At least one schema item/collection pair of objects
- At least one response item (e.g. HomeResponse)
- At least one requestBodies item (e.g. addItem)

See exercise document for examples

Leveraging OpenAPI Components



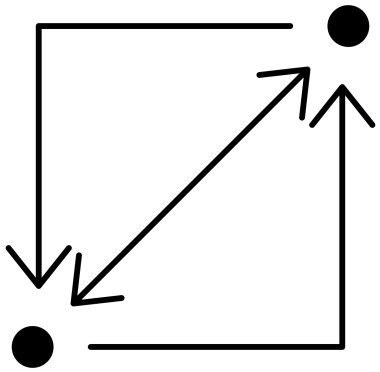
- The OpenAPI Components Section
  - Organize your OpenAPI definition
  - Improve the consistency of your API design
- Parameters, Schemas, Responses, & RequestBodies
  - Parameters for inline queries
  - Schemas for shared objects (item/collection)
  - Responses for general use responses (e.g. errors)
  - requestBodies for POST/PUT requests (e.g. addItem)



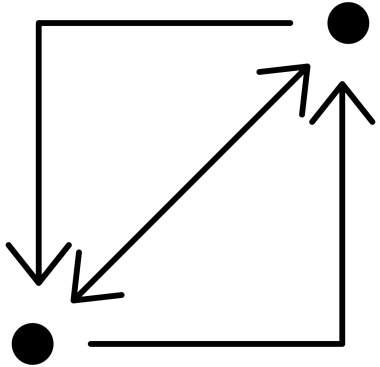
Defining OpenAPI Paths

Defining OpenAPI Paths

- The OpenAPI Paths Section
- Elements of a Path Definition
- Exercise: Document Your OpenAPI Paths



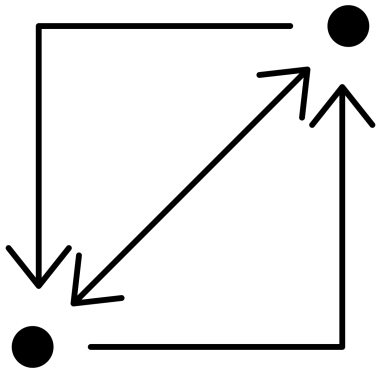
The OpenAPI Paths Section



- Paths are the 'heart' of OpenAPI documents
- One top-level entry for each resource/URL
- One or more HTTP methods for each resource
- One or more `parameters`, `requestBodies`, `responses`

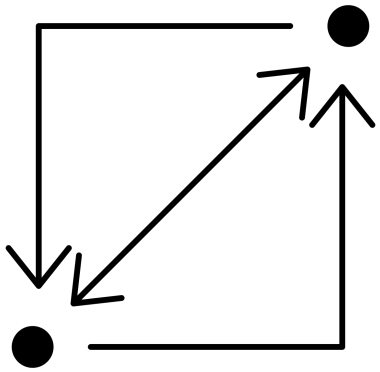
Elements of a Path Definition

```yaml
### home ###
/:
  get:
    operationId: home
    summary: Use this to retrieve the home page for the onboarding API
    tags:
    - onboarding

    parameters:
      - $ref: "#/components/parameters/eTag"

    responses:
      '200':
        $ref: "#/components/responses/reply"

      default:
        $ref: "#/components/responses/error"
```



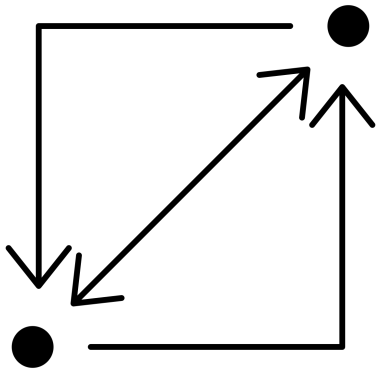
Elements of a Path Definition

```yaml
### start ###
/start:
  post:
    operationId: startOnboarding
    summary: Use this to start process of onboarding a customer
    tags:
    - onboarding

    parameters:
      - $ref: "#/components/parameters/ifNoneMatch"

    requestBody:
      $ref: "#/components/requestBodies/start"

    responses:
      '201':
        $ref: "#/components/responses/created"

      default:
        $ref: "#/components/responses/error"
```

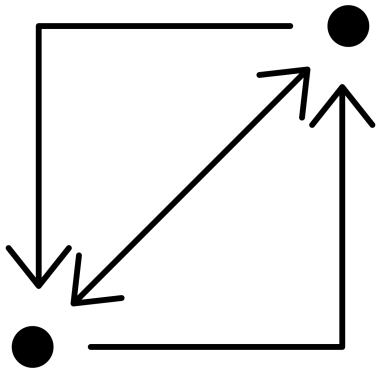


Elements of a Path Definition

```yaml
### work in progress record ###
/wip/{identifier}:
  get:
    operationId: wipItem
    summary: Use this to return a single onboarding record
    tags:
    - onboarding

    parameters:
      - $ref: "#/components/parameters/identifier"
      - $ref: "#/components/parameters/eTag"

    responses:
      '200':
        $ref: "#/components/responses/reply"

      default:
        $ref: "#/components/responses/error"
```



Exercise: Document your OpenAPI Paths

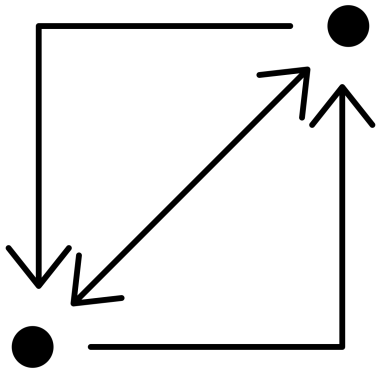


Add path elements to your OpenAPI document

- One or more URLs (/, /collection, /collection/{identifier}, etc.)
- One or more methods per URL (GET, PUT, DELETE, etc.)
- Add `parameters`, `requestBody`, and `response` as needed

Update your `components` section when appropriate

Defining OpenAPI Paths : Review



- The OpenAPI Paths Section
  - `paths` are the 'heart' of OpenAPI documents
- Elements of a Path Definition
  - Top-level resource url
  - One or more HTTP methods
  - `parameters`, `requestBody`, and `response`

Mocking and Testing Your API Design

Mocking and Testing Your API Design

- What is a Mock?
- Generating API Mocks with OpenAPI
- Writing Simple Request Tests (SRTs) for OpenAPI
- Exercise: Test your Mock API

Mocking and Testing Your API Design

"I unwittingly discovered the concept of a Mock out of necessity!"

-- Gerard Meszaros

Generating API Mocks with OpenAPI

Mock testing is a software testing technique that involves creating simulated or artificial components, known as mocks, to mimic the behavior of real components in a system.

Generating API Mocks with OpenAPI

simple-api

API OAS3.0

Details Integrations

API Auto Mocking

Add New Integrations

Owner amundsen

Created By mamund

Last Saved 8:02:56 pm Nov 20, 2024

en in Explore

CTFLOW Sync Aa

Export

Code Form Preview

```
  required: true
  content:
    application/json:
      schema:
        type: object
        properties:
          status:
            type: string
            example: "active"
responses:
  '200':
    description: "Status updated successfully."
  '404':
    description: "Person not found."
tags:
  - Item

e:
t:
summary: "Retrieve home view"
description: "Retrieve the home view for the person service (safe operation)."
operationId: goHome
responses:
  '200':
    description: "Home view retrieved successfully."
tags:
  - Home

ents:
mas:
rson:
type: object
properties:
  id:
    type: string
    description: "Unique identifier for the person."
  name:
    type: string
    description: "Name of the person."
  email:
    type: string
    description: "Email address of the person."
```

Person Service API

1.0.0 OAS 3.0

API derived from the provided ALPS document for managing person resources.

Servers

https://api.example.com - Productio...

Collection

GET /collection List persons

POST /collection Create a new person

GET /collection/filter Filter persons

Item

Writing Simple Request Tests for OpenAPI

- SRTs are a great way to validate endpoints
- Use **`curl`** to execute simple requests against your API
- "Eyeball" the results to make sure it works as expected
  - Does the request fail?
  - Does the body "look ok?"

```
# person api test requests
# 2020-02 mamund

http://localhost:8181/
http://localhost:8181/list/
http://localhost:8181/filter?status=active
http://localhost:8181/ -X POST -d id=q1w2e3r4&status=pending&email=test@example.org
http://localhost:8181/q1w2e3r4 -X PUT -d givenName=Mike&familyName=Mork&telephone=123-456-7890
http://localhost:8181/status/q1w2e3r4 -X PATCH -d status=active
http://localhost:8181/q1w2e3r4 -X DELETE

# EOF
```

Writing Simple Request Tests for OpenAPI

- SRTs are a great way to validate endpoints
- Use **`curl`** to execute simple requests against your API
- "Eyeball" the results to make sure it works as expected
  - Does the request fail?
  - Does the body "look ok?"

```
# person api test requests
# 2020-02 mamund

http://localhost:8181/
http://localhost:8181/list/
http://localhost:8181/filter?status=active
http://localhost:8181/ -X POST -d id=q1w2e3r4&status=pendin
http://localhost:8181/q1w2e3r4 -X PUT -d givenName=Mike&fam
http://localhost:8181/status/q1w2e3r4 -X PATCH -d status=ac
http://localhost:8181/q1w2e3r4 -X DELETE

# EOF
```

```
http://localhost:8181/23456 -X GET
Time:  1584657412556 : localhost:8181/23456 : GET : {}
{
  "error": [
    {
      "type": "error",
      "title": "SimpleStorage: [api]",
      "detail": "Not Found [23456]",
      "status": "400",
      "instance": "http://localhost:8181/23456"
    }
  ]
}
```

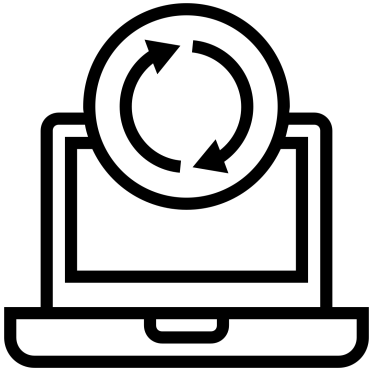
Exercise: Test Your Mock API

- Turn on mocking for your OpenAPI in Swagger Editor
- Edit your copy of the SRT list to include your mock URL
- Run the SRT script to confirm your mosk is up and running

Optionally, add more URLs to your SRT script to create additional tests for your API mock.

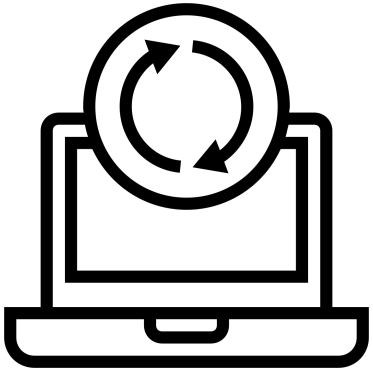
Mocking & Testing Your API Design : Review

- What is a Mock?
  - Simulated test object/target
- Generating API Mocks with OpenAPI
  - Mocks make it easy to test
- Writing Simple Request Tests (SRTs) for OpenAPI
  - SRTs make testing easy



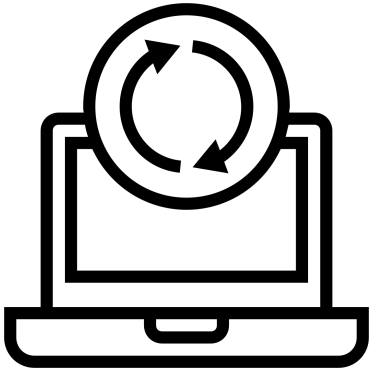
Modifying Your API in Production

Modifying Your API in Production



- Beyond Versioning
- Three Rules for Modifying Production APIs
- Testing and Deploying Modified APIs
- Exercise: Verify the Safety of your API Modifications

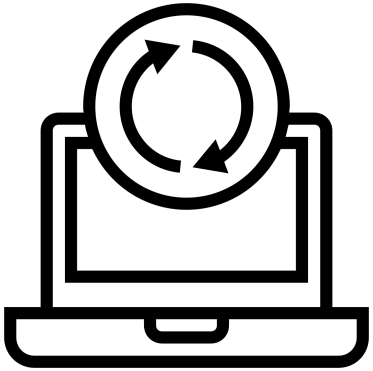
Beyond Versioning



- Dealing with Change
  - Change is a feature, not a bug
- First, Do No Harm
  - You may not know who your consumers are
- Forking Your API
  - Multiple forks mean multiple versions in production

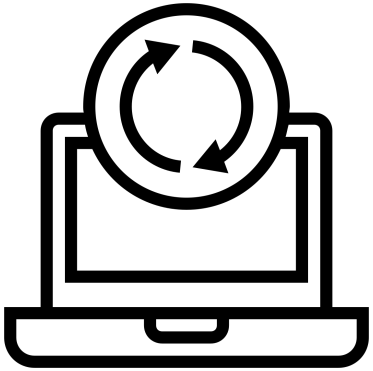
"First, do no harm" is the Hippocratic Oath of APIs

Three Rules for Modifying Production APIs



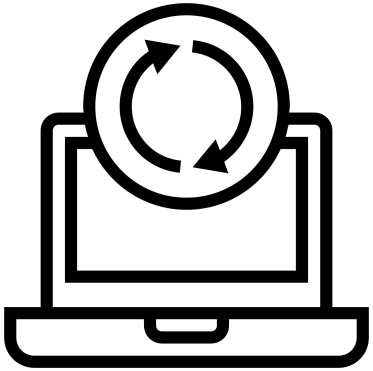
- Take Nothing Away
  - Features are frozen
- Don't Redefine Anything
  - Meaning is frozen
- Make All Additions Optional
  - No new REQUIRED inputs

Testing and Deploying Modified APIs



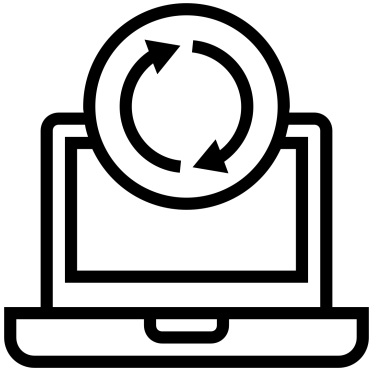
- Use all existing tests
  - Do not replace/rewrite them
- Add new tests for the modifications
  - Be sure to test all the new features
- Run all tests (existing & new) everytime

Testing and Deploying Modified APIs



- Support Reversible deployments
  - You MUST be able to "undo" a production update
- Use the Salesforce approach (forking)
  - Support side-by-side releases
- Use the AWS approach (overwriting)
  - Only release in-place compatible updates

Verify the Safety of Your API Modifications

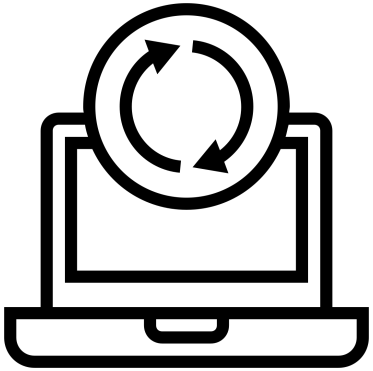


Modify your existing OpenAPI design and re-run your tests

- Start with your existing OpenAPI document
- Make sure your SRT script passes all tests
- Add a new URL resource, method, or parameter to your API
- Update your SRT script to test the new design element
- Re-run the tests (both existing and new) to confirm safe modifications

See exercise for additional details

Modifying Your API in Production : Review



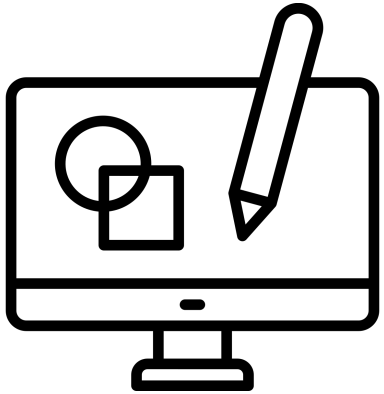
- Beyond Versioning
  - First, do no harm
- Three Rules for Modifying Production APIs
  - Take nothing away, don't redefine, additions are optional
- Testing and Deploying Modified APIs
  - Don't replace tests, support reversible releases, salesforce vs AWS

Review

Designing APIs with OpenAPI : Review

- The Power of API Design
- API Design Assets
- Prototyping Your API Design with OpenAPI
- Leveraging OpenAPI Components
- Defining OpenAPI Paths
- Mocking and Testing Your API Design
- Modifying Your API in Production

The Power of API Design : Review

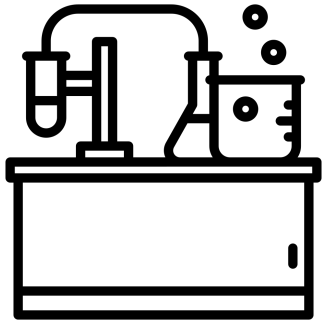


- API-First
  - First class citizen
- Norman's Lifecycle
  - Everything is a loop
- Three Pillars of API Design
  - Business, audience, design features

API Design Assets : Review

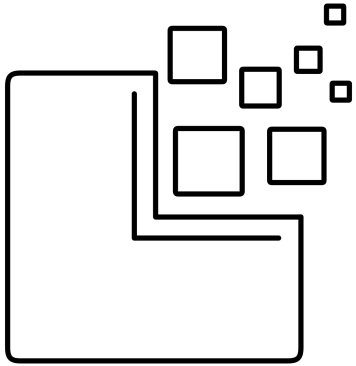
- API Story
  - Shared, accessible, repeatable
- API Diagram
  - Display the interactions
- API Vocabulary
  - The "magical words" of the API

Prototyping Your API Design : Review



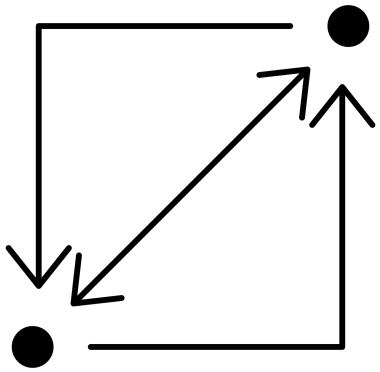
- What is an API Prototype?
  - A preliminary model
- Why Use Prototypes?
  - Prototypes are made to be tested
- OpenAPI Info Section
  - Version, title, description

Leveraging OpenAPI Components



- The OpenAPI Components Section
  - Organize your OpenAPI definition
- Parameters, Schemas, Responses, & RequestBodies
  - Parameters for inline queries
  - Schemas for shared objects (item/collection)
  - Responses for general use responses (e.g. errors)
  - requestBodies for POST/PUT requests (e.g. addItem)

Defining OpenAPI Paths : Review

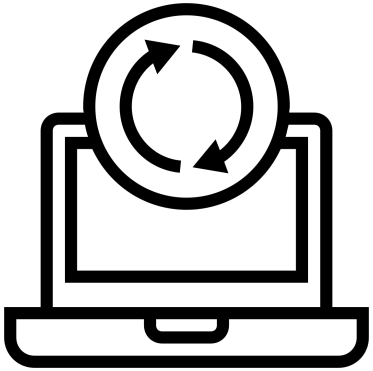


- The OpenAPI Paths Section
  - `paths` are the 'heart' of OpenAPI documents
- Elements of a Path Definition
  - Top-level resource url
  - One or more HTTP methods
  - `parameters`, `requestBody`, and `response`

Mocking & Testing Your API Design : Review

- Generating API Mocks with OpenAPI
  - Mocks make it easy to test
- Writing Simple Request Tests (SRTs) for OpenAPI
  - SRTs make testing easy

Modifying Your API in Production : Review



- Beyond Versioning
  - First, do no harm
- Three Rules for Modifying Production APIs
  - Take nothing away, don't redefine, additions are optional
- Testing and Deploying Modified APIs
  - Don't replace tests, support reversible releases, salesforce vs AWS

Designing APIs with OpenAPI

Mike Amundsen
@mamund