Good API Design

Zdenek “Z” Nemec
z@goodapi.co

GOOD API

Independent API Consulting

Adam Kliment

Follow @ntmlk

Erik Wilde

Follow @dret

Zdenek "Z" Nemec

Follow @zdne

adidas · angelcam · Deutsche Post DHL Group · LEI Global Legal Entity Identifier Foundation · KONICA MINOLTA · mystay · WINDING TREE · ytica

Good API Design

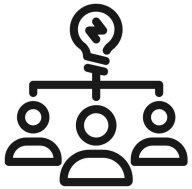
What it takes to design a good API?

API-first

The Three Pillars

Three Pillars of Digital Transformation API-first

Business



Organization

Technology

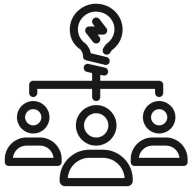
Business

APIs are the products that you create, manage, sell, measure, recombine, and retire.

API First means that if a product in your organization does not have an API, it essentially does not exist.

APIs are the connective fabric.

Organization



No matter how good your service or product is, if it is delivered through a bad API, this substantially diminishes the quality of the product.

Teams embrace the fact that APIs is their main deliverables.

Technology

Everything that gets created or consumed in the organization is based on APIs.

The quality of APIs greatly matter for how easy it is to create and consume company products.

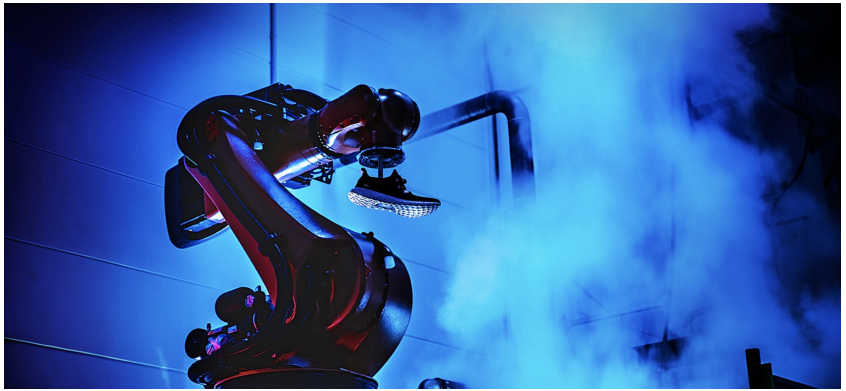
APIs change must not create expensive and time-consuming ripple effects through a chain of API dependencies.

APIs are no longer just about the technology

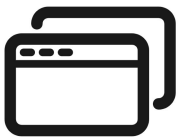
API-first in Practice

Everything produced and consumed in the organization has an API

Sports vs. Tech Company



Where API Comes First



API as first UI

API Before Implementation

API Documentation

The API Product

The API Product

API-as-a-Product

Direct API monetization.

Where possible, the company product is offered directly through a paid API.

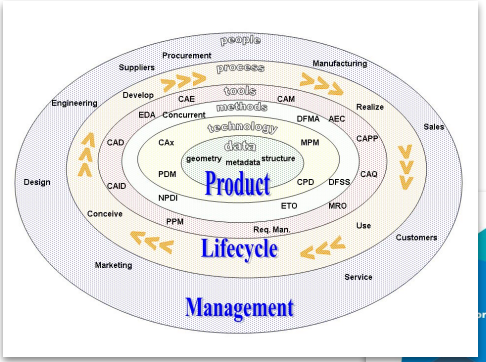
Product as API

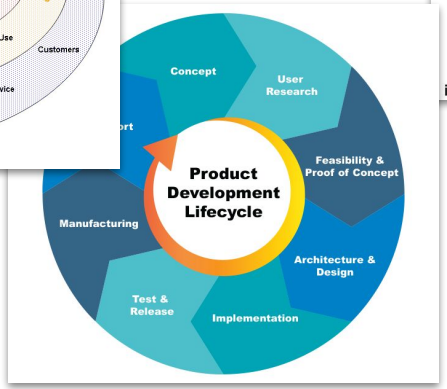
Indirect “monetization”.

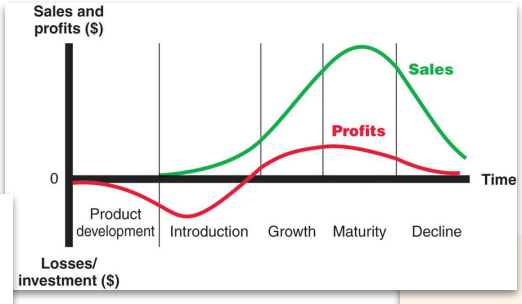
Each product is an API and used through its API. APIs are contributing indirectly to the bottomline through increasing stickiness, driving revenue-generating activities, leveraging partner channels, increasing efficiency or leading innovation.

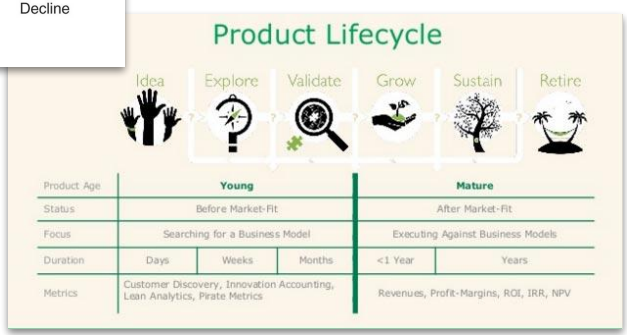
Design-first

Treat API Development as Product Development









Design-first

API Before Implementation

As with any good product, the work on API product starts with hypothesis, verification and design–prototyping.

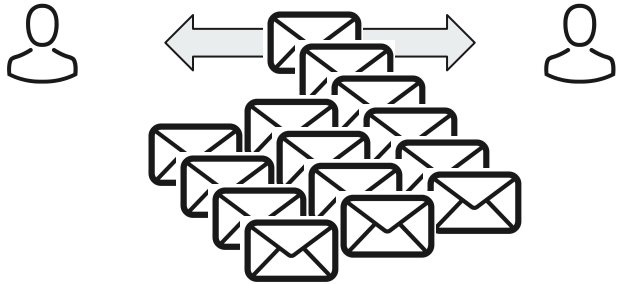
Introduction to API Design

API Design

The Email Story

RE: RE: RE: RE: RE: RE: The API We Want

Have you ever tried to discuss what an API should look like over a plain email?



API Design

Good API Design

API design is API architectural style decision together with a set of additional design constraints applied on a specific domain, business, domain, organizational and technical requirements.

API Design

Good API Design is Verified

- Prototype
- Share
- Verify
  - Stakeholders
  - Users
  - API Developers

API Description

API Description

API Design is formalized using one of the API Description formats

- API Blueprint
- RAML
- Swagger
- OpenAPI Specification 2.0 (fka Swagger)
- OpenAPI Specification 3.0
- AsyncAPI Specification
- GraphQL Schema
- Avro Schema
- Protobuf

API Description

OpenAPI Specification 2.0 (formerly Swagger)

```yaml
swagger: '2.0'
info:
  description: This is a simple API
  version: 1.0.0
  title: Simple Inventory API
  # put the contact info for your development or API team
  contact:
    email: you@your-company.com

  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html

# tags are used for organizing operations
tags:
- name: admins
  description: Secured Admin-only calls
- name: developers
  description: Operations available to regular developers

paths:
  /inventory:
    get:
      tags:
      - developers
      summary: searches inventory
      operationId: searchInventory
      description: |
        By passing in the appropriate options, you can search for
        available inventory in the system
      produces:
      - application/json
```

API Description

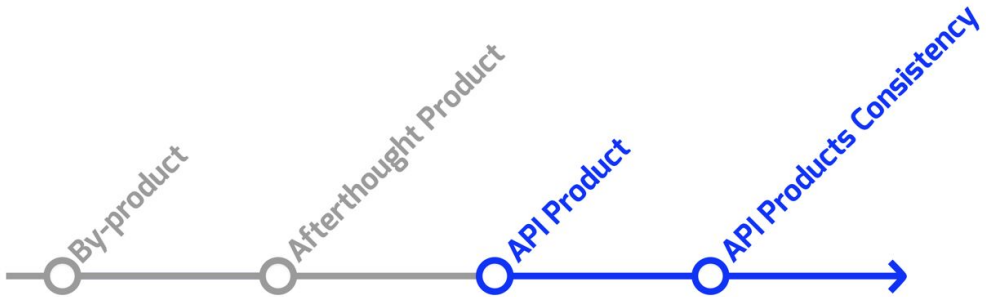
API Description must be:

- Human & machine readable
- Accessible
- Understood by Stakeholders, Developers, API Consumers, Tech writers, DevOPS, Support team
- Versioned in VCS
- Treated as Product Requirements

Approved API design becomes the contract

Design Consistency

API Design Maturity



Consistency

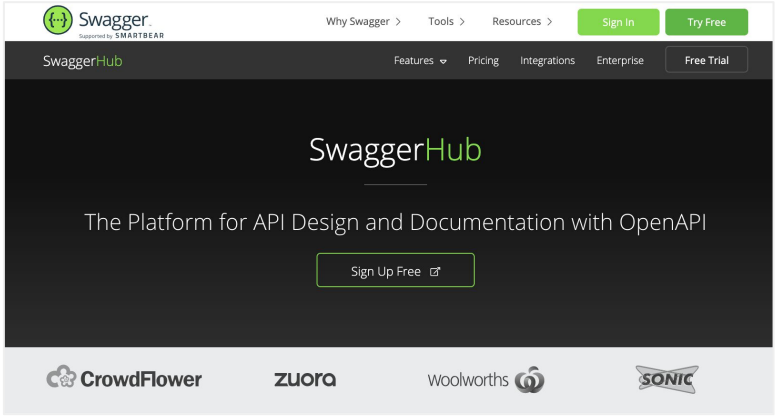
Consistency

Consistency is a win-win for both API consumers and providers:

- Reduces complexity
- Encourages re-use
- Enables upsell
- Reduces time-to-market
- Reduces maintenance
- Enables scaling

How to achieve consistency

Visibility



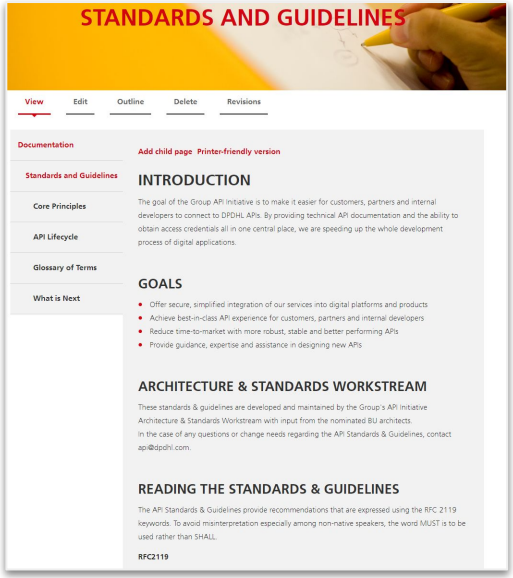
ORACLE + apiary

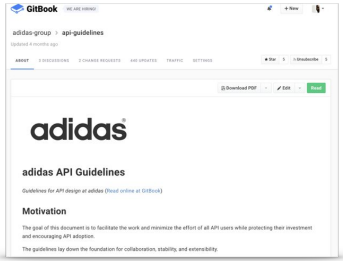
Stoplight

API Repository

Organization-wide API repository – register with every API design in the organization.

Standardization

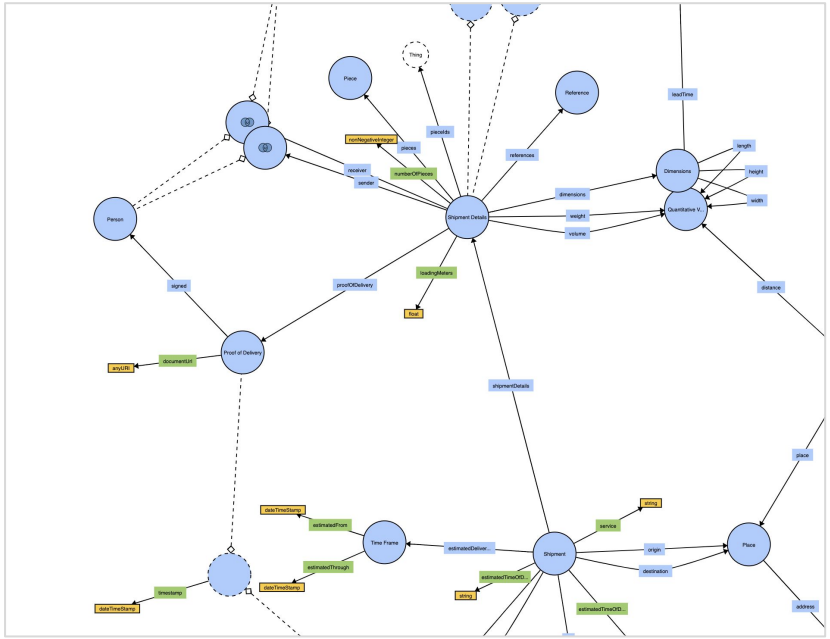




Vocabulary Harmonization & Reconciliation

https://vocabulary.dhl.com/logistics

Vocabulary aligned with schema.org and GS1



Automated Consistency Checks

Automated Consistency Checks

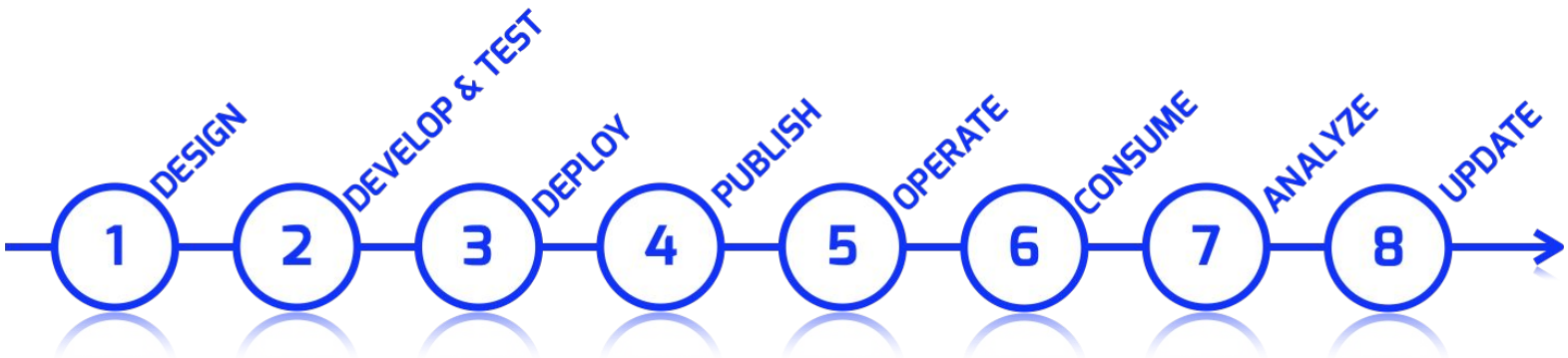
Automated API guidelines consistency checks using tools like Apiary Style Guide or Stoplight Spectral.

ORACLE + apiary

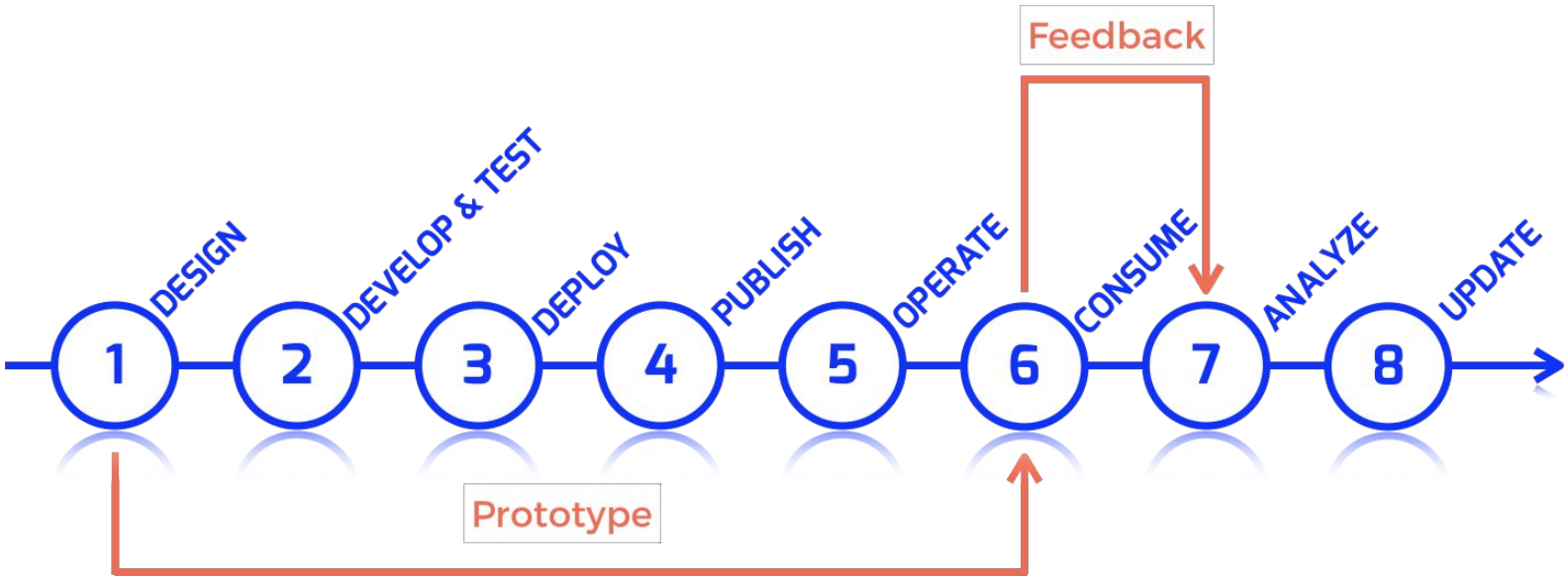
Spectral

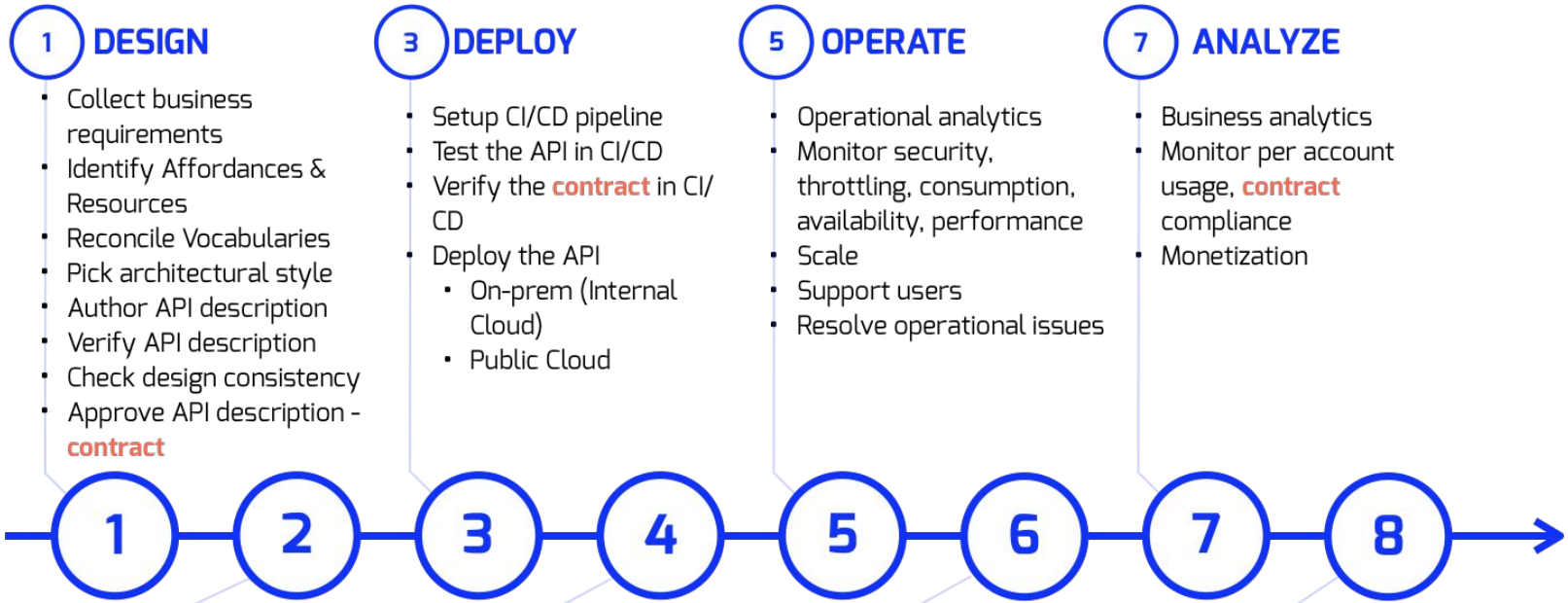
API Lifecycle

Waterfall Lifecycle



Agile API Lifecycle




1 DESIGN
Collect business requirements
Identify Affordances & Resources
Reconcile Vocabularies
Pick architectural style
Author API description
Verify API description
Check design consistency
Approve API description - contract
3 DEPLOY
Setup CI/CD pipeline
Test the API in CI/CD
Verify the contract in CI/CD
Deploy the API
On-prem (Internal Cloud)
Public Cloud
5 OPERATE
Operational analytics
Monitor security, throttling, consumption, availability, performance
Scale
Support users
Resolve operational issues
7 ANALYZE
Business analytics
Monitor per account usage, contract compliance
Monetization
1
2
3
4
5
6
7
8
2 DEVELOP & TEST
Implement the API surface
Test the API locally
Verify the contract locally
4 PUBLISH
Expose the API internally or externally based on the contract
Security, throttling, consumption setup
Operational & Business analytics setup
6 CONSUME
Publish API on Dev portal
API documentation based on the contract
Obtain API credentials (keys, tokens)
Experiment with API
Use the API
User support
8 UPDATE
Review the design
Collect new business requirements
Update the API description
Approve the updated API description - new contract

API Design

API Design Phase

1. **Collect business requirements, use cases**
   Greenfield vs. Brownfield scenario, verification of hypothesis and business alignment

2. **Understand, Reconcile and Define Vocabularies**
   Domain-driven design, explore vocabularies and data models

3. **Pick architectural style, decide on the protocol**
   Role of the API Architect

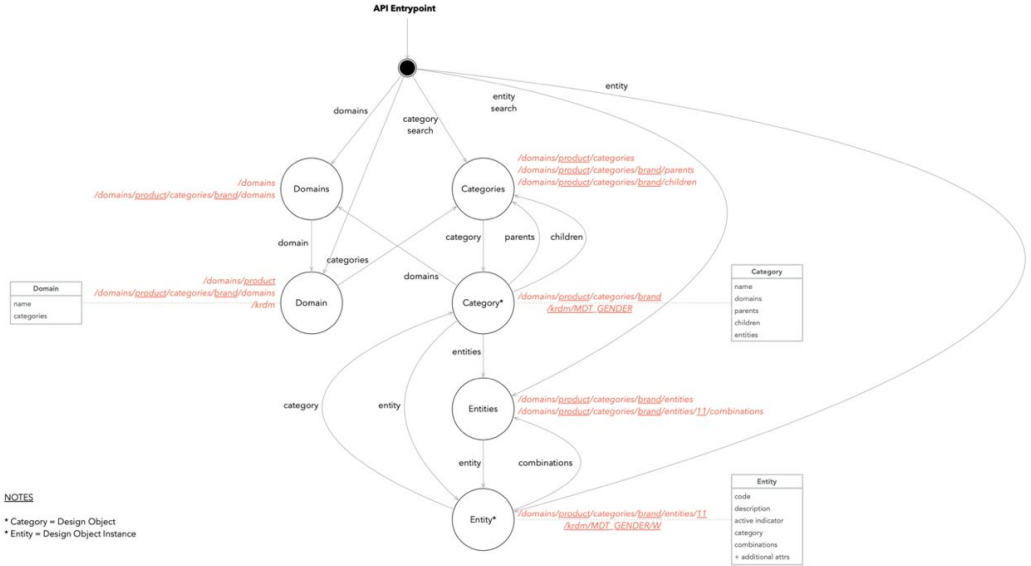
API Architectural Styles Workshop

4. **Design the API**
   Based on the style and protocol selection

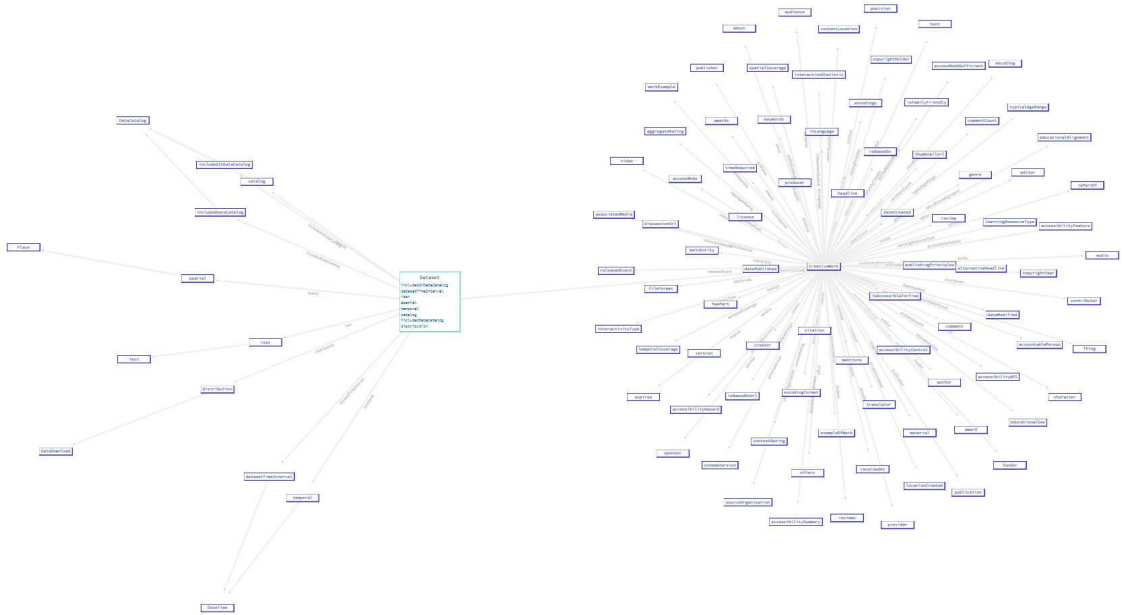
Design API – Web APIs

1. **Identify Affordances & Resources**
2. **Draw Finite State Machine Diagram**
3. **Decide on the Message Formats (Resource Representations)**
4. **Author API Description**
    a. Define resources & actions
    b. Define models (e.g. using supermodel.io)
    c. Add authentication mechanism

Web API Automaton



Supermodel Data Model



Design API – GraphQL APIs

1. **Identify core queries**
2. **Identify mutations**
3. **Define models**

   (e.g. using supermodel.io)
4. **Author GraphQL Schema**

API Design Phase

5. **Check API Design consistency**
   Style-guides & design patterns, design governance

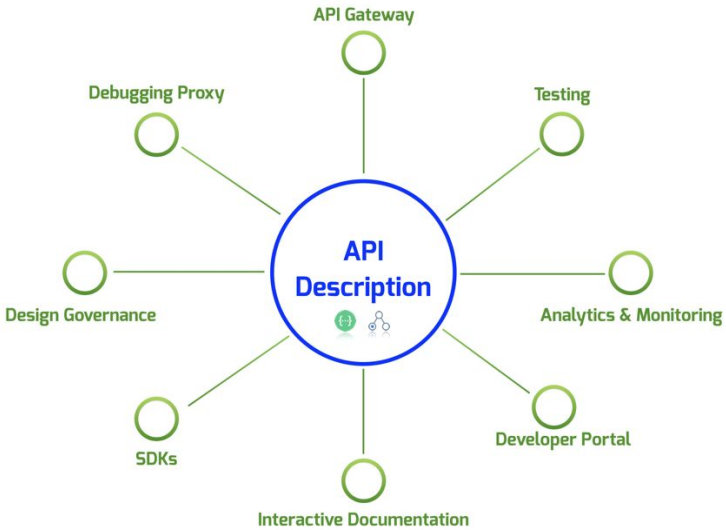
6. **Verify API Description**
   Developers, clients & stakeholders; Use mock and documentation driven by the API description

7. **Approve the API Description–contract**
   Binding for every party in API lifecycle

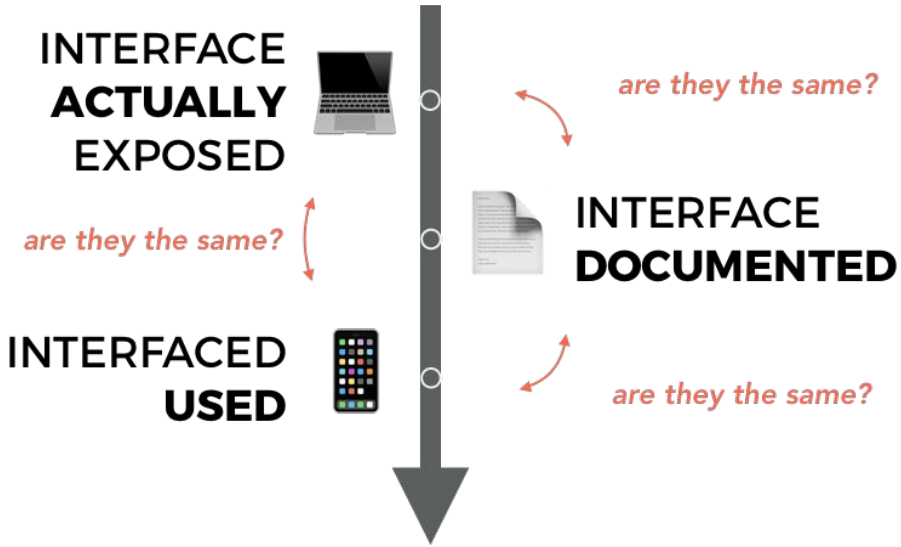
8. **Evolve the design–Update Phase**

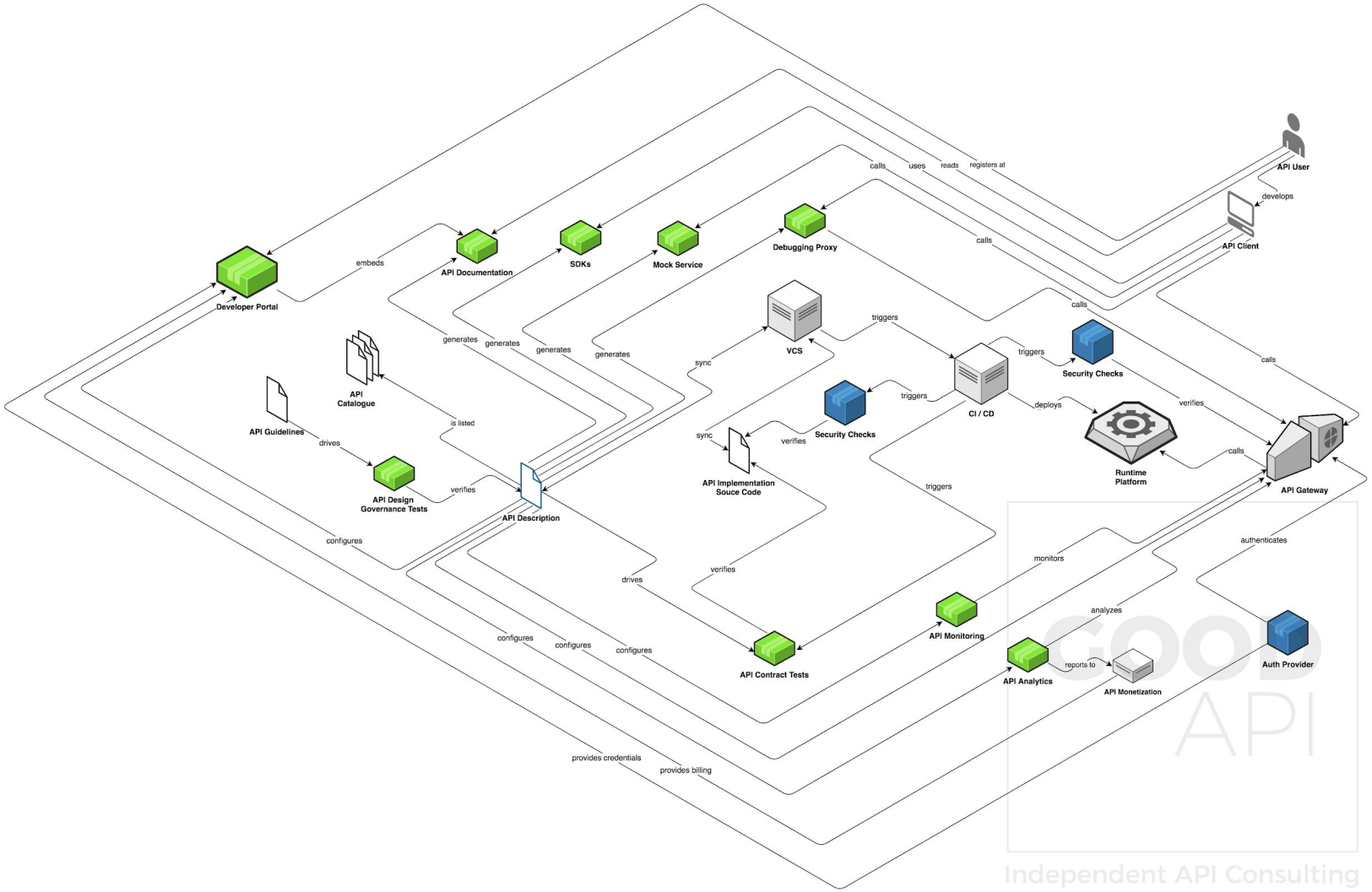
Contract-driven API Lifecycle

API Lifecycle Must be Contract-driven



Keeping Things in Sync is Hard




API User
develops
API Client
calls
uses
reads
registers at
Developer Portal
embeds
API Documentation
SDKs
Mock Service
Debugging Proxy
calls
calls
generates
generates
generates
generates
API Catalogue
API Guidelines
is listed
drives
API Design Governance Tests
verifies
API Description
configures
sync
VCS
triggers
triggers
Security Checks
calls
triggers
CI / CD
deploys
verifies
Runtime Platform
calls
API Gateway
sync
API Implementation Souce Code
verifies
Security Checks
triggers
drives
verifies
configures
configures
configures
API Contract Tests
monitors
API Monitoring
analyzes
API Analytics
reports to
API Monetization
authenticates
Auth Provider
provides credentials
provides billing
GOOD
API
Independent API Consulting

Evolving the API (design)

The first rule of Fight Club

Never break clients

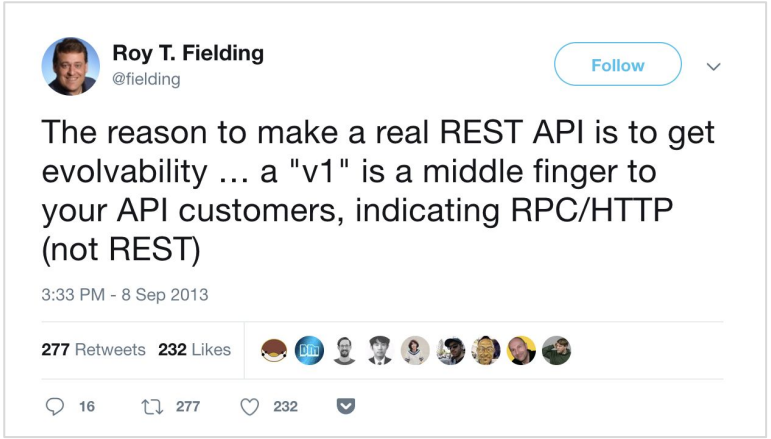
Any modification to an existing API MUST avoid breaking changes and MUST maintain backward compatibility.

Rules for extending

1. You MUST NOT take anything **away**
2. You MUST NOT change **processing rules**
3. You MUST NOT make optional things **required**
4. Anything you add MUST be **optional**

Identifier Stability

URI is an identifier



Backward incompatible changes

New Resource Variant

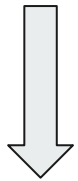
T0

Parameters `first` and `last` optional

T+1

Parameter `first` required

/greeting?first=John&last=Appleseed



New Resource Variant

/named-greeting?first=John&last=Appleseed

Hands-on

1. Collect business requirements, use cases

We are working for a florist company Florist LLC. selling physical products–flowers. As we want to transform our business for the digital era we want to build an order API to provide easy integration for our partners.

For the start we would like our API to serve following use cases:

As a user–customer of the florist company I want to be able to:

1. Retrieve list of my orders
2. Retrieve details about one of my orders, including its items
3. Check status of an order
4. Place a new order
5. Cancel order that has not been yet shipped
6. Return an order

These use cases were already verified with the partners-potential users.

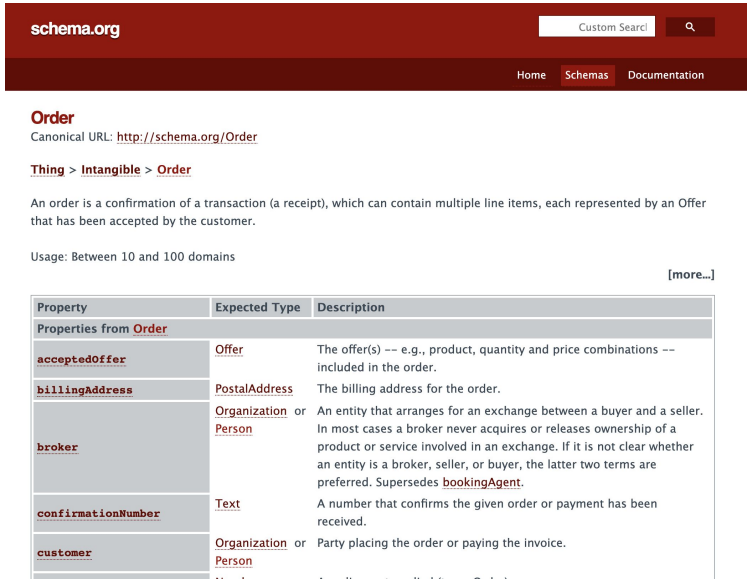
2. Reconcile and Define Vocabularies

Our florist company does not have its own formal vocabulary. But after a quick check of schema.org we are confident we can use the schema.org Order concept without any need for extensions.

Our company order management systems defines the following statuses (schema.org OrderStatus) of the order:

- **Pending (Created) ***
- Confirmed
- **Cancelled ***
- Shipped
- Received
- **Returned ***

Our API will allow its users to create/change only those statuses marked with (*)



3. Pick architectural style, decide on the protocol

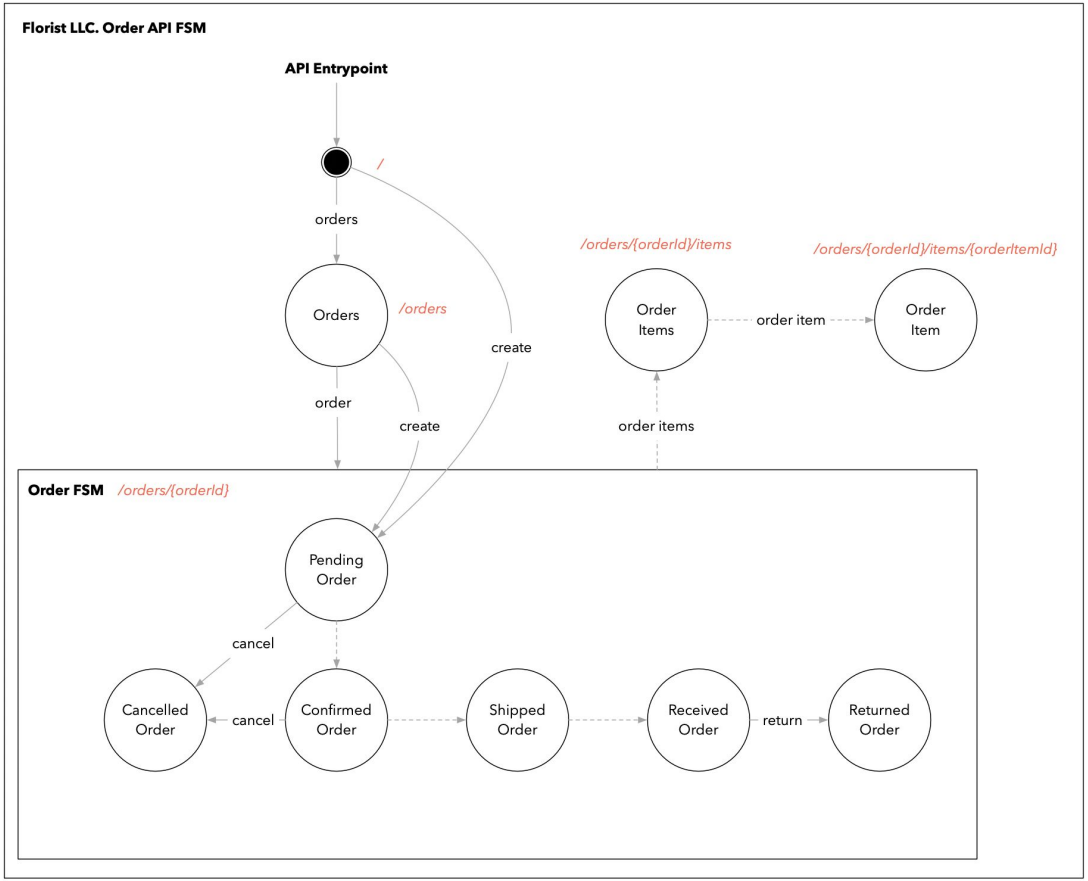
Based on our business requirements and organizational constraints our API architect has decided to create the API as a proper REST API using the HTTP1.1 protocol.

4. Design API – Web APIs

With use cases, domain model and architectural style given, we are going to design the REST API.

Draw FSM

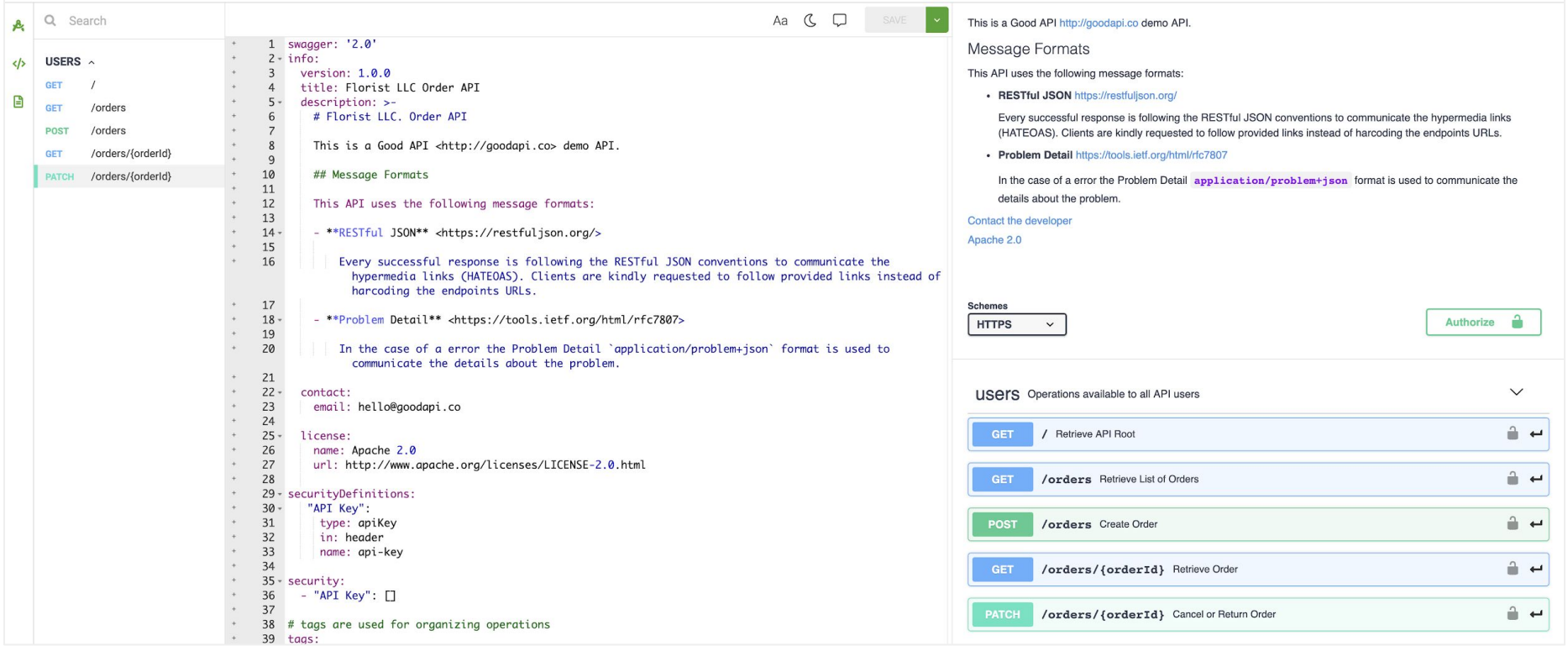
Your task is to identify the resources in the API and relations between them and draw a finite state machine (FSM – automaton) diagram of the API.



Florist LLC. Order API Automaton

Author API Description

https://app.swaggerhub.com/apis/goodapi/demo-order-api/1.0.0#/



Florist LLC. Order API API Description

5. Check API Design consistency

If the organization has API Standards and guidelines this is the time where we should check the compliance of the authored API description.

Ideally, the checks are automated.

6. Verify API Description

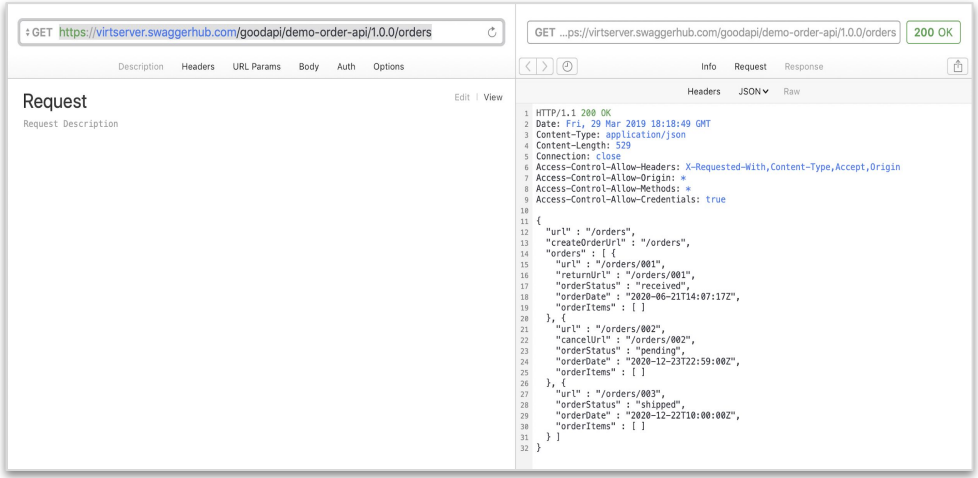
Using the tools driven by the authored API description we are going to distribute the API prototype documentation and experiment with the stub implementation (mock server).

For a good design, it is important the everybody involved in the API lifecycle is engaged including the potential customers.

Call the API

Your task is to explore the Florist LLC. Order API documentation and make an actual call to the API! Use your preferred tool to make a call (cURL, Postman, Paw). Try to call different endpoints and observe the response.

The URL of the API documentation is:
https://app.swaggerhub.com/apis/goodapi/demo-order-api/1.0.0#/
or:
https://bit.ly/2I1s0Bc



Florist LLC. Order API Mock Interaction

7. Approve the API Description, establish Contract

With verifications done, approved API Descriptions becomes the contract binding all parties in the API lifecycle.

7. Evolve, Update or Retire

As time goes a new requirements - use cases are requested:

As a user–customer of the florist company I want to be able to:

1. Back-order an already received order
2. Retrieve trimmed-down list of my orders without all the order line information

Risk: Potentially backward-incompatible change

Independent API Consulting

Icons by IconMark from the Noun Project

GOOD API

Independent API Consulting