Contract-first API development

using the OpenAPI Specification (fka Swagger)

Ian Zelikman

@izcoder

Dave Forgac

@tylerdave

Questions

BetterAPIs.com

https://github.com/tylerdave/OpenAPI-Tutorial/

Updates

Get repo updates

```
git pull
```

Reprovision VM

```
vagrant suspend
vagrant reload --provision
```

Log in

```
vagrant ssh
```

Backup Plan

http://editor.swagger.io/

Background

REST

REST

~~Standard~~

REST

~~Standard~~

Architectural Style

REST

~~Standard~~

Architectural Style

HTTP w/ Constraints

REST

~~Standard~~

Architectural Style

HTTP w/ Constraints

REST-inspired HTTP APIs

API Contract

API Contract

Client ↔ Provider

API Contract

Client ↔ Provider

Interface Specification

API Contract

Client ↔ Provider

Interface Specification

SLA, ToS, Limits, Pricing, etc.

JSON

JSON

JavaScript Object Notation

JSON

JavaScript Object Notation

```
{
    "things": [
        "foo",
        "bar"
    ],
    "message": "Hello, World!"
}
```

JSON Schema

JSON Schema

```
{
   "title": "Example Schema",
   "type": "object",
   "properties": {
      "displayName": {
         "type": "string"
      },
      "age": {
         "description": "Age in years",
         "type": "integer",
         "minimum": 0
      }
   },
   "required": ["firstName", "lastName"]
}
```

YAML

YAML

Serialization format

YAML

Serialization format

(More) human-readable

YAML

Serialization format

(More) human-readable

Superset of JSON

YAML

Serialization format

(More) human-readable

Superset of JSON

Language Support

YAML Basics - Lists

```
---
colors:
  - red
  - green
  - blue
...
```

YAML Basics - Dictionaries

```
---
session:
  title: Contract-First API Development
  type: tutorial
...
```

YAML Basics - Spanning

```
---
description: |
  This is a long description
  using a pipe
  which will preserve newlines.
description2: >
  This is a long desciption using > which
  will ignore
  new
  lines.
...
```

YAML Basics - Nesting

```
---
session:
  name: Contract-First API Development
  type: tutorial
  topics:
    - apis
    - openapi specification
    - swagger
  languages: ['java', 'nodejs', 'python']
  description: >
    A really useful tutorial during which you'll
    learn about API specifications and stuff.
...
```

Compare

JSON

```
"Talk": {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer"
    },
    "title": {
      "minLength": 1,
      "type": "string",
      "maxLength": 144
    }
  }
}
```

YAML

```
Talk:
  type: object
  properties:
    id:
      type: integer
    title:
      type: string
      minLength: 1
      maxLength: 144
```

API Definitions

API Definitions

WSDL / WADL

API Definitions

WSDL / WADL

Swagger -> OpenAPI Spec

API Definitions

WSDL / WADL

Swagger -> OpenAPI Spec

API Blueprint

API Definitions

WSDL / WADL

Swagger -> OpenAPI Spec

API Blueprint

RAML

OpenAPI Spec

OpenAPI Spec

Structure

OpenAPI Spec

Structure

History

OpenAPI Spec

Structure

History

Future

OpenAPI 3.0

OpenAPI 3.0

Coming Soon

OpenAPI 3.0

Coming Soon

Tooling to Follow

This Tutorial

Goals

Goals

OpenAPI Spec

Goals

OpenAPI Spec

Testing

Goals

OpenAPI Spec

Testing

Mock

Goals

OpenAPI Spec

Testing

Mock

Basic Implementation

Goals

OpenAPI Spec

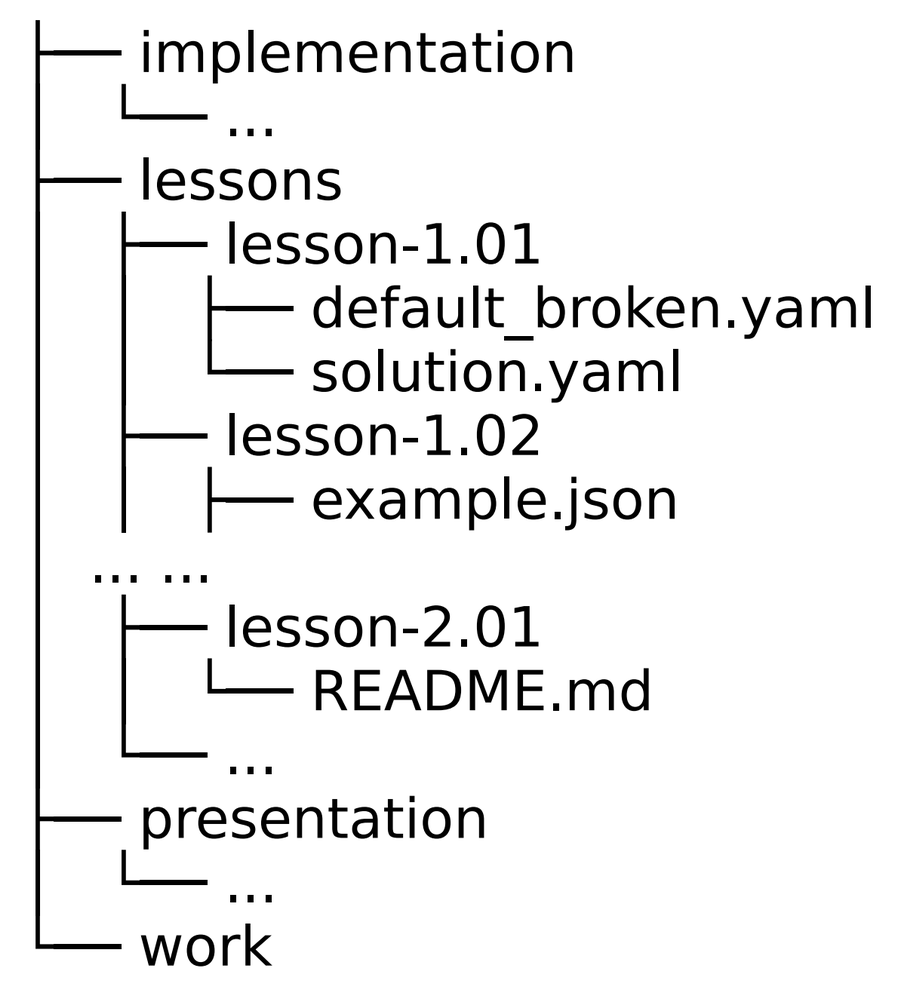
Testing

Mock

Basic Implementation

Documentation

Repo Layout


implementation
...
lessons
lesson-1.01
default_broken.yaml
solution.yaml
lesson-1.02
example.json
... ...
lesson-2.01
README.md
...
presentation
...
work

Synced Folder

Repo dir on host

mapped to

/home/ubuntu/tutorial-repo/ **on VM**

Instructions

- Work in `work` directory
  - Via editor on host machine
  - Or via editor in VM terminal
- Save `betterapis.yml`
- Run validator within VM:
  - `swagger validate tutorial-repo/work/betterapis.yml`

Lesson Solutions

Done?

- Compare with contents of `lessons/lesson-x.xx/solution.xxx`

Stuck?

- Copy `lessons/lesson-x.xx/solution.xxx` to `work`

Tutorial

Lesson 1.01: Setup

Goals

- Explore the environment.
- Look at some Open API example specs and exercise the tools we will use.

Lesson 1.01: Setup

Tooling

- Swagger editor:

```
http://localhost:8000/
```

- Validator:

```
swagger validate tutorial-repo/lessons/lesson-1.02/solution.yaml
```

Exercise Instructions

- Load several examples from the swagger editor, review them.
- Import the broken examples from lesson-1.01 directory. Try fixing the errors.

Solution 1.01

Notes

- Got familiar with basic OpenAPI Spec structure

Lesson 1.02: Hello, World!

Goals

- Building a first (basic) spec.

YAML Example

JSON Example

Lesson 1.02: Hello, World!

Exercise Instructions

- Build an API for a conference called `betterapis`
- Include metadata as shown in the example
- Paths are empty for now

Solution 1.02

Notes

- This solution might be a bit different than yours in regards to the metadata.
- Valid Spec!

Lesson 1.03: Pets

Goals

- Get familiar with defining paths.

Lesson 1.03: Pets

Basic Path

```
/pets:
  get:
    summary: Get a list of pets
    description: Retrieve a list of pets
    responses:
      201:
        description: OK
```

Lesson 1.03: Pets

Exercise Instructions

- Add two paths to the API: `/talks` `/speakers` .
- Both paths only support GET and only return status code 200.

Solution 1.03

Notes

- We defined the very basic fields and objects needed for a valid path.

Lesson 1.04: Registration

Goals

- Learn to define complex operations on the API.

Lesson 1.04: Registration

Paths, Actions

```
paths:
    /pets:
      post:
        summary: Add pet to DB
        description: Results in new pet information added to the DB
        parameters:
          - name: pet
            in: body
            description: Pet details
            schema:
              required: [name, status]
              properties:
                name:
                  type: string
                  description: The pet name
```

Paths, Actions (contd.)

```
responses:
  201:
    description: Created new pet in the database
    schema:
      required: [pet-id]
      properties:
        pet-id:
          type: number
          description: Unique Id for the pet in the system
```

Lesson 1.04: Registration

Exercise Instructions

- Add actions to support speaker registration and talk submission
- You are free to define the speaker and talk objects as you like as long as you define a unique id in both and exercise defining more than one basic type for the object properties.

Solution 1.04

Notes

- Additional properties: `readOnly`, `format`, `pattern`

Lesson 1.05: The Minimalist API

Goals

- Reusing definitions.
- Learn more in depth about action objects and request parameters.

Lesson 1.05: The Minimalist API

Path Parameter

```
/pets/{pet-id}:
  parameters:
    pet-id:
      name: pet-id
      in: path
      description: Pet identifier
      type: number
      required: true
```

Parameter Reuse

```
/pets/{pet-id}:
  parameters:
    - $ref: '#/parameters/pet-id'
    ...
parameters:
  pet-id:
    name: pet-id
    in: path
    description: Pet identifier
    type: number
    required: true
```

Definitions Reuse

```
...
    schema:
      $ref: '#/definitions/Pet'
...
definitions:
  Pet:
    type: object
    required: [name, status]
    properties:
  ...
  Pets:
    type: array
    items:
      $ref: "#/definitions/Pet"
```

Lesson 1.05: The Minimalist API

Exercise Instructions

1. Refactor your API to use `Talk` and `Speaker` objects. Define `Talks` and `Speakers` objects based on the previous and update the responses from `/speakers` and `/talks` paths.
2. Add a two new paths `/speakers/{speaker-id}` and `/talks/{talk-id}`. Define all the CRUD operations for them and use parameter definition outside of the action for path parameter.

Solution 1.05

Notes

- Time saving with definition. More readable.
- Example response in the solution

Lesson 1.06: Responses

Goals

- Learn more about parameter definition via pagination
- Learn How to define reusable responses
- Default responses

Lesson 1.06: Responses

Pagination

```
    parameters:
     - $ref: '#/parameters/page-size'
     - $ref: '#/parameters/page-number'
    ...
parameters:
  page-size:
    name: page-size
    in: query
    description: Number of items
    type: integer
    format: int32
    minimum: 1
    maximum: 100
    multipleOf: 10
    default: 10
```

Lesson 1.06: Responses

Response Definition

```
responses:
  ServerErrorResponse:
    description: Server error during request.
    schema:
      $ref: "#/definitions/Error"
definitions:
  Error:
    properties:
      code:
        type: integer
      message:
        type: string
```

Lesson 1.06: Responses

Default Response

```
/pets/{pet-id}/
  delete:
    responses:
      ...
      default:
        $ref: '#/responses/UnknownResponse'
responses:
  UnknownResponse:
    description: This response is not yet documented by this API.
```

Lesson 1.06: Responses

Exercise Instructions

- Add pagination to the `/talks` and `/speakers` paths. Pagination should be included by at least two parameters: `page-size`, `page-number`.
- Add the following responses to all paths: 400, 500, default.

Solution 1.06

Notes

- Functionality complete API

Lesson 1.07: Secure Your APIs

Goals

- Learn the different security schemas supported.
- Global vs. local security via file upload definition example.

Lesson 1.07: Secure Your APIs

Basic Auth

```
securityDefinitions:
  type: basic
```

Lesson 1.07: Secure Your APIs

API Key

```
securityDefinitions:
  "type": "apiKey",
  "name": "api_key",
  "in": "header"
```

Lesson 1.07: Secure Your APIs

OAuth2

```
securityDefinitions:
  OauthSecurity:
    type: oauth2
    flow: accessCode
    authorizationUrl: 'https://oauth.swagger.io.com/authorization'
    tokenUrl: 'https://oauth.swagger.io/token'
    scopes:
      admin: Admin scope
      user: User scope
```

```
security:
  - OauthSecurity:
    - user
```

Lesson 1.07: Secure Your APIs

File Upload

```
paths:
  /pets/{pet-id}/picture:
    parameters:
          - $ref: '#/parameters/pet-id'
    post:
      description: Admin operation to upload a pet picture
      operationId: UploadPicture
      security:
        - OauthSecurity:
          - admin
      consumes:
        - multipart/form-data
      parameters:
        - name: picture
          in: formData
```

Lesson 1.07: Secure Your APIs

Exercise Instructions

- Define a security scheme for your API. Use Oauth2.
- Add a new path to be able to upload speaker resume and secure it using admin role.

Solution 1.07

Notes

- Security representation in the editor
- Header in responses

Lesson 1.08: Doc the Docs

Goals

- Learn additional points on spec documentation

Lesson 1.08: Doc the Docs

OperationId

```
/pets:
    get:
      operationId: GetPets
```

Description GFM

```
/pets:
    get:
      description: ## Retrieve multiple pet objects.
                   For example:
                   - pet1
                   - pet2
```

Lesson 1.08: Doc the Docs

Tags

```
paths:
  /pets:
      get:
        tags:
          - pet
  ...
tags:
  name: pet
  description: Pet operations
```

Lesson 1.08: Doc the Docs

Exercise Instructions

- Update for API with more information on the operations description, using GFM.
- Add tags and operationIds to all your operations

Solution 1.08

Notes

- Tags in the editor

Lesson 1.09: Can We Split This?

Goals

- Learn how to support not having all the API in one flat file

Lesson 1.09: Can We Split This?

Reference External Files

```
/pets:
    get:
      summary: Get a list of pets
      description: Retrieve a list of pets
      operationId: GetPets
      parameters:
        - $ref: 'parameters.yaml#/page-size'
        - $ref: 'parameters.yaml#/page-number'
```

Lesson 1.09: Can We Split This?

parameters.yaml

```
Parameters:
  page-size:
    name: page-size
    in: query
    description: Number of items
    type: integer
    format: int32
    minimum: 1
    maximum: 100
    multipleOf: 10
    default: 10
```

Lesson 1.09: Can We Split This?

Serving External Files

Lesson 1.09: Can We Split This?

Exercise Instructions

- Split your API spec. The proposed scheme is to have separate file for definitions, parameters and responses. You can consider other split strategies.

Solution 1.09

Notes

- A better-organized specification

Part 1 Recap

- 1.01: Setup
- 1.02: Hello, World!
- 1.03: Pets
- 1.04: Registration
- 1.05: The Minimalist API
- 1.06: Responses
- 1.07: Secure Your APIs
- 1.08: Doc the Docs
- 1.09: Can We Split This?

Break

Contract-first API development

using the OpenAPI Specification (fka Swagger)

Part 2

What do we get?

What do we get?

Benefits

Benefits

Documentation

Benefits

Documentation

Mocking

Benefits

Documentation

Mocking

Testing

Benefits

Documentation

Mocking

Testing

Code Generation

Benefits

Documentation

Mocking

Testing

Code Generation

Code Generation

Code Generators

Code Generators

Servers

Code Generators

Servers

Clients

Code Generators

Servers

Clients

Documentation

Swagger-Codegen

Swagger-Codegen

Via Swagger editor

Calls to `https://generator.swagger.io/`

Swagger-Codegen

Via Swagger editor

Calls to `https://generator.swagger.io/`

Via CLI

`http://swagger.io/swagger-codegen/`

Integrated Frameworks

Integrated Frameworks

Swagger Inflector (Java)

Integrated Frameworks

Swagger Inflector (Java)

swagger-node (Node.js)

Integrated Frameworks

Swagger Inflector (Java)

swagger-node (Node.js)

Connexion (Python)

Lesson 2.01: Code Generation

Goals

- Server/Client Code from Spec

Generate Server

Generate Client

Exercise Instructions

- Generate server & client side code with your favorite option provided by the code generator.
- (bonus) Update server side code so that the `/talks` and `/speakers` paths return empty list on GET. Use the methods provided by the client code in order test the responses from the server.

Solution 2.01

Notes

- Experimented with code generated

Connexion

Connexion

Connexion

Python + Flask

Connexion

Python + Flask

Spec As Configuration

Connexion

Python + Flask

Spec As Configuration

Routing, Validation, etc.

Connexion

Python + Flask

Spec As Configuration

Routing, Validation, etc.

Explicit Routing

Explicit Routing

Explicit Function Name

```
paths:
  /hello_world:
    post:
      operationId: myapp.api.hello_world
```

Explicit Routing

Explicit Function Name

```
paths:
  /hello_world:
    post:
      operationId: myapp.api.hello_world
```

Separate Controller Name

```
paths:
  /hello_world:
    post:
      x-swagger-router-controller: myapp.api
      operationId: hello_world
```

Automatic Routing

Automatic Routing

```
from connexion.resolver import RestyResolver
app = connexion.FlaskApp(__name__)
app.add_api('swagger.yaml', resolver=RestyResolver('api'))
```

Automatic Route Resolution

```
paths:
  /:
    get:
      # Implied operationId: api.get
  /foo:
    get:
      # Implied operationId: api.foo.search
    post:
      # Implied operationId: api.foo.post
  '/foo/{id}':
    get:
      # Implied operationId: api.foo.get
    put:
      # Implied operationId: api.foo.put
    copy:
      # Implied operationId: api.foo.copy
    delete:
      # Implied operationId: api.foo.delete
```

Request Validation

Request Validation

JSON Schema

Request Validation

JSON Schema

Required parameters

Request Validation

JSON Schema

Required parameters

Types and Formats

Request Validation

JSON Schema

Required parameters

Types and Formats

Custom Validators

Request Validation

JSON Schema

Required parameters

Types and Formats

Custom Validators

HTTP 400 w/ Details

Response Handling

Response Handling

Serialization

Response Handling

Serialization

JSON Encoder

Response Handling

Serialization

JSON Encoder

Validation Optional

Response Handling

Serialization

JSON Encoder

Validation Optional

Custom Validators

Security

Security

OAuth 2 via Spec

Security

OAuth 2 via Spec

DIY

API Key

Basic Auth

Other Features

Other Features

Swagger UI

Other Features

Swagger UI

Swagger JSON

Other Features

Swagger UI

Swagger JSON

Flask Integration

Other Features

Swagger UI

Swagger JSON

Flask Integration

Lesson 2.02: Run the API

Goals

- Python/Flask
- Connexion

Connexion Implementation

Lesson 2.02: Run the API

Exercise Instructions

- Run the the betterapis application using the connexion implementation
  - Activate virtualenv: `workon tutorial`
  - Run app: `python -m betterapis`
- Register two speakers and submit a talk for each one.
  - Use HTTP POSTs via Postman, curl, et al.
- Request speaker list to verify data persisted.

Solution 2.02

Notes

- Populated data

Lesson 2.03: Mock Server

Goals

- Run mock server for client to experiment with the API

Lesson 2.03: Mock Server

EXAMPLES

```
responses:
    200:
      description: Returns a specific talk
      schema:
        $ref: '#/definitions/Pet'
      examples:
        application/json:
          {
            id: 12345,
            name: "pythagoras",
            status: "Adopted"
          }
```

Lesson 2.03: Mock Server

Usage

```
connexion run betterapis.yaml --mock=all -v
```

Lesson 2.03: Mock Server

Exercise Instructions

- Update responses to have examples
- Run and test your mock server

Solution 2.03

Notes

- Can mock with any spec

Lesson 2.04: Test with Dredd

Goals

- Learn how the spec connects tests and implementation

Lesson 2.04: Test with Dredd

Installation

- Using npm
- Provided in the VM

Lesson 2.04: Test with Dredd

Usage

```
Dredd init
```

```
? Location of the API description document
   tutorial-repo/implementation/betterapis/specs/betterapis.yaml
? Command to start API backend server e.g. (bundle exec rails server)
? URL of tested API endpoint http://127.0.0.1:8080
? Programming language of hooks python
? Do you want to use Apiary test inspector? No
? Dredd is best served with Continuous Integration.
   Create CircleCI config for Dredd? No
Configuration saved to dredd.yml
Run test now, with:
  $ dredd
```

How Dredd works

- In request: `x-example` or `default`
- In response: format matching

Parameter Example

```
pet-id:
  name: pet-id
  in: path
  description: Pet identifier
  type: number
  required: true
  x-example: 42
```

Lesson 2.04: Test with Dredd

Exercise Instructions

- Update all the GET actions so that they can be tested using Dredd.
- Initialize dredd and run `dredd --method GET` command in order to verify that tests are passing. (Use ids from data you initialized in the application in previous lesson)

Solution 2.04

Notes

- 4 tests passed and 6 skipped.

Lesson 2.05: New Features

Goals

- Development cycle for new feature

Lesson 2.05: New Features

Flow

- Update the spec
- Run tests -> Fail
- Update the code
- Run tests -> Success

New Feature Demo

Lesson 2.05: New Features

Exercise Instructions

- Add a new feature to the application to support reviews. Besides a unique `id` the review object should reference the `talk_id` it refers to and `details`.

Solution 2.05

Notes

- Connecting it all together

Lesson 2.06: Documentation

Goals

- Generating automatic documentation for clients

Lesson 2.06: Documentation

Tooling

- Connexion: HTML, Console
- swagger-ui
- Many others

Connexion Docs Demo

Exercise Instructions

- Register a new speaker and submit a talk using the connexion UI.
- Use the UI to also update and delete the talk and the speaker.

Solution 2.06

Notes

- Easy to distribute documentation

Part 2 Recap

- 2.01: Code Generation
- 2.02: Run the API
- 2.03: Mock Server
- 2.04: Test with Dredd
- 2.05: New Features
- 2.06: Documentation

More Resources

BetterAPIs.com

Thank You!

Thank You!

Questions?

Ian
ian.zelikman@gmail.com
@izcoder

Dave
dave@forgac.com
@tylerdave