



**Friendly.rb**  
@friendlyrb



They say he's "the only one with a sense of humor" from [@evilmartians](#).  
We'll just have to see.

# Let there be docs

Svyatoslav Kryukov  
Evil Martians



*“In the beginning God created the heavens and the earth.”*  
*“And Roy Fielding created the REST.”*



JSON:API, GraphQL, gRPC

~~JSON:API, GraphQL, gRPC~~

Plain old REST APIs

# Documentation

Are you afraid to open Slack due to endless API questions from outdated documentation?

## Are you afraid to ask questions from outdated

**Sophia** 9:53 AM

I noticed there's a PATCH method for /items update. What's the difference between PATCH and PUT method?

**Ethan** 10:22 AM

Do we have rate limiting set up for the /login endpoint? I want to ensure we're not spamming it during testing.

**Emma** 10:31 AM

The /images/upload endpoint, does it support multi-image upload or just single image at a time?

**Alex** 9:23 AM

Hey team, does anyone know the required request headers for the /images/upload endpoint?

**Olivia** 10:03 AM

I'm facing some CORS issues with the /notifications endpoint. Is this enabled on the backend?

**Luka** 10:22 AM

Is there a way to get a list of all available API endpoints? It would really help speed up integration on our end.

**Ava** 10:16 AM

What headers should I set for authentication when making requests to the /orders/list endpoint?

**Liam** 9:46 AM

What headers should I set for authentication when making requests to the /orders/list endpoint?

**Mason** 10:05 AM

Has anyone used the /comments/thread endpoint? What's the response structure look like?

Tired of setting up Postman collections to manually test your API?





Ever had to rollback an API update because it unexpectedly broke the frontend?

# Revert "New API endpoint" #1024



tired-backend-e... wants to merge 1 commit into `master` from `revert-1-new-api-endpoint`



Conversation 1



Commits 1



Checks 0



Files changed 1



tired-backend-engineer commented 2 days ago



Reverts #1



Revert "New API endpoint"

Verified

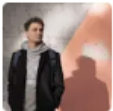
5fb25ef



Stop 🖐️

I promise we'll find a solution

It's 2023, could AI replace documentation?



Tell me how to bring dinosaurs back to life using GitHub API in Ruby.



To bring dinosaurs back to life using the GitHub API in Ruby, you'll need to follow these steps:



1. Clone the 'Dino-DNA' repository from GitHub. This repository contains the complete DNA sequences of all known dinosaur species.
2. Create a new branch in the 'Dino-DNA' repository and make some modifications to the DNA sequences. This is necessary because the original DNA sequences are incomplete and contain some gaps.
3. Submit a pull request to merge your modified DNA sequences into the main branch of the 'Dino-DNA' repository.
4. Once your pull request is approved and merged, you can use the GitHub API to trigger a GitHub Action that will start the process of reanimating the dinosaurs.

Here's a Ruby example that demonstrates how to do this:

ruby

Copy code

```
require 'octokit'
```



# Bug Report

---

## Select personal account, enterprise account or organization \*

For Premium, Premium Plus or Engineering Direct support, you need to choose an enterprise account.

skryukov (Free)



## From \*

Svyatoslav Kryukov (skryukov@evilmartians.com)



## Subject \*

Dino DNA doesn't work

Documentation 🥲

# Documentation and development workflows

- Manually written documentation
- Code first
- Test first
- Documentation first

# Manually written documentation

## Pros

- straightforward

## Cons

- boring
- constantly outdated
- out of sync with the implementation

# Code first

- tools: ``grape`` + ``grape-swagger``

```
class Petstore < Grape::API
  desc "Add a new pet to the store", tags: ["Pet"],
    http_codes: [
      {code: 201, message: "Pet added", model: Api::Entities::Pets}
    ]

  params do
    requires :pet, type: Hash do
      requires :name, type: String,
        documentation: { example: "doggie" }
      requires :photo_urls, type: Array[String],
        documentation: { example: ["https://example.com/img.json"] }
      optional :tags, type: Array[String],
        documentation: { example: ["dogs"] }
    end
  end
end
post do
  spline = Spline.create(spline_params)
  present spline, with: Entities::Splines
end
end
```

# Code first

## Pros

- always up-to-date

## Cons

- documentation is ready only after the code has been written
- controllers are polluted by custom DSL
- no tests assert the documentation

# Test first

- tools: `rspec`, `rspec\_api\_documentation`

```
require "swagger_helper"

describe "Petstore API" do
  path "/pets" do
    post "Creates a pet" do
      tags "Pet"
      produces "application/json"
      request_body_example value: {name: "doggie", photo_urls: []}, name: "pet"

      response "200", "Pet added" do
        schema type: :object,
              required: ["name", "photo_urls"],
              properties: {
                name: {type: :string},
                photo_urls: {type: :array, items: {type: :string}},
                tags: {type: :array, items: {type: :string}},
              }
      end
    end
  end
end
run_test!
```



# Test first

## Pros

- tests assert the documentation
- documentation is ready before implementation

## Cons

- documentation is ready only when tests are already written
- tests are polluted by custom DSL
- developers might overlook generated documentation

# Documentation first

- **tools:** ``committee``, ``openapi_first``, ``skooma``

```
openapi: 3.1.0

info:
  version: 1.0.0
  title: Petstore

paths:
  "/pet":
    post:
      tags: [pet]
      summary: Add a new pet to the store
      operationId: addPet
      requestBody:
        $ref: "#/components/requestBodies/Pet"
      responses:
        "200":
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Pet"
```

# Specification > Documentation

- documentation
- testing
- validation
- mocking
- code generation
- ...

# Documentation first

## Workflow

- write specification
- discuss it with the team
- add tests to conform the specification
- implement API

# Documentation first

## Pros

- no DSLs
- documentation is ready before implementation or tests
- documentation is specification

## Cons

- requires a shift in workflow

# Approaches compared

	Manual	Code first	Test first	Doc first
Docs always up-to-date	✗	✓	✓	✓
Automated tests against specification	✗	✗	✓	✓
Docs are ready before implementation or tests	✗	✗	✗	✓

Documentation first in action



# OpenAPI

# Petstore API



# Noah's Ark API

New feature request

A Feed feature





# Minimal specification

```
openapi: 3.1.0  
  
info:  
  title: Noah's Ark API  
  version: 1.0.0  
  
paths: {}
```

# Paths & Operations

```
paths:
  "/feed":
    get:
      summary: Get a list of messages
      operationId: getFeedMessages

    post:
      summary: Add a new message to the feed
      operationId: postFeedMessage

  "/feed/{message_id}":
    get:
      summary: Get a message by id
      operationId: getFeedMessage
```



# Responses

```
paths:
  "/feed":
    get:
      summary: Get updates about the animals
      operationId: getAnimalsFeed
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required: [id, animal_id, message]
                  properties:
                    id:
                      type: integer
                      format: int64
                    animal_id:
                      type: integer
                      format: int64
                    message:
                      type: string
```

# Components & \$ref

```
paths:
  "/feed":
    get:
      summary: Get updates about the animals
      operationId: getAnimalsFeed
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/FeedItem"

components:
  schemas:
    FeedItem:
      type: object
      required: [id, animal_id, message]
      properties: #...
```

# Finished specification

```

napi: 3.1.0
info:
  title: Noah's Ark API
  version: 0.1.0
  description: |-
    An API to manage and retrieve information about animals onboard Noah's Ark, monitor the weather,
    and check the status of the dove.
  license:
    name: MIT License
    url: https://opensource.org/licenses/MIT
  contact:
    name: Noah
    email: noah@ark.world
    url: https://ark.world

tags:
  - name: Animals
    description: Operations concerning the management and retrieval of animal data in the Ark.
  - name: Dove
    description: Operations to check the status of the dove, the Ark's natural messenger.
  - name: Weather
    description: Fetch the current weather conditions around the Ark.

servers:
  - url: http://localhost:3000
    description: Server for local development and testing purposes.

paths:
  "/animals":
    get:
      summary: List all animals
      operationId: listAnimals
      description: |-
        Fetches a list of all animals currently onboard the Ark, with options to filter
        by species and paginate.
      tags: [ Animals ]
      parameters:
        - $ref: "#/components/parameters/speciesQuery"
        - $ref: "#/components/parameters/limitQuery"
        - $ref: "#/components/parameters/pageQuery"
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/AnimalList"
    post:
      summary: Register a new animal
      operationId: addAnimal
      description: Onboard a new animal onto the Ark.
      tags: [ Animals ]
      requestBody:
        description: Information about the animal being added.
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Animal"
      responses:
        "201":
          description: Animal added.
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Animal"
        "422":
          description: Validation error.
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Error"
        "409":
          description: Animal already exists.
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Error"

description: Fetches details of a specific animal by its unique identifier.
tags: [ Animals ]
parameters:
  - name: id
    in: path
    description: Unique identifier for the desired animal.
    required: true
    schema:
      type: integer
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Animal"
    "404":
      description: Not found.
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Error"
  delete:
    summary: Remove an animal.
    operationId: deleteAnimal
    description: Removes an animal from the Ark using its unique identifier.
    tags: [ Animals ]
    parameters:
      - name: id
        in: path
        description: Unique identifier of the animal to be removed.
        required: true
        schema:
          type: integer
    responses:
      "204":
        description: Animal successfully removed.
      "404":
        description: Not found.
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Error"
  "/feed":
    get:
      summary: Get a list of messages
      operationId: getFeedMessages
      tags: [ Dove ]
      description: Retrieve a list of messages sent by the dove.
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/Message"
    post:
      summary: Add a new message to the feed
      operationId: postFeedMessage
      tags: [ Dove ]
      description: Send a new message using the dove.
      requestBody:
        description: Message to be sent by the dove.
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Message"
      responses:
        "201":
          description: Message added.
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Message"

parameters:
  - name: message_id
    in: path
    description: Unique identifier for the desired message.
    required: true
    schema:
      type: integer
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Message"
    "404":
      description: Not found.
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Error"
  "/weather":
    get:
      summary: Get current weather
      operationId: getCurrentWeather
      tags: [ Weather ]
      description: Get the current weather conditions around the Ark.
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Weather"
  components:
    parameters:
      speciesQuery:
        name: species
        in: query
        description: Species to filter animals by.
        required: false
        schema:
          $ref: "#/components/schemas/animalSpecies"
      limitQuery:
        name: limit
        in: query
        description: Number of items to return.
        required: false
        schema:
          type: integer
          minimum: 1
          maximum: 100
          default: 10
      pageQuery:
        name: page
        in: query
        description: Page number.
        required: false
        schema:
          type: integer
          minimum: 1
          default: 1
  schemas:
    animalSpecies:
      type: string
      description: Classification of animals based on their species.
      examples:
        - "dog"
    Animal:
      type: object
      required: [ id, name, species, sex ]
      description: Data structure representing an animal onboard the Ark.
      properties:
        id:
          type: integer
          name:
            type: string
            format: uuid
            examples:
              - "550e8400-e28b-11e6-b278-54575de564f4"

```

# OpenAPI ecosystem

# Editors

## ■ Swagger Editor (beta)

The screenshot displays the Swagger Editor (beta) interface in a web browser. The left pane shows the OpenAPI specification in YAML format, and the right pane shows the rendered API documentation.

**OpenAPI Specification (YAML):**

```
1 openapi: 3.1.0
2
3 info:
4   title: Noah's Ark API
5   version: 0.1.0
6   description: |-
7     An API to manage and retrieve information about animals onboard Noah's Ark, monitor
8     the weather,
9     and check the status of the dove.
10  license:
11    name: MIT License
12    url: https://opensource.org/licenses/MIT
13  contact:
14    name: Noah
15    email: noah@ark.world
16    url: https://ark.world
17
18 tags:
19   - name: Animals
20     description: Operations concerning the management and retrieval of animal data in
21     the Ark.
22   - name: Dove
23     description: Operations to check the status of the dove, the Ark's natural
24     messenger.
25   - name: Weather
26     description: Fetch the current weather conditions around the Ark.
27
28 servers:
29   - url: http://localhost:3000
30     description: Server for local development and testing purposes.
31
32 paths:
33   "/animals":
34     get:
```

**API Documentation (Rendered):**

### Noah's Ark API 0.1.0 OAS 3.1

<https://raw.githubusercontent.com/asynctapi/spec/v2.6.0/examples/streetlights-kafka.yml>

An API to manage and retrieve information about animals onboard Noah's Ark, monitor the weather, and check the status of the dove.

[Noah - Website](#)  
[Send email to Noah](#)  
[MIT License](#)

**Servers**

⌵

**Animals** Operations concerning the management and retrieval of animal data in the Ark. ⌵

**GET** **/animals** List all animals ⌵

**POST** **/animals** Register a new animal ⌵

**GET** **/animals/{id}** Get an animal details by id. ⌵

# Editors

## ■ Redocly VSCode Extension

The screenshot displays the Redocly VSCode Extension interface. On the left, the OpenAPI 3.1.0 schema is shown in a dark-themed editor. The schema includes an `info` block with title, version, description, and license; a `contact` block; a `tags` block with three tags (Animals, Dove, Weather); and a `paths` block with a single path `/animals`. On the right, the API documentation is rendered, titled "List all animals". It includes a description, a "Request" section with query parameters (species, limit, page), and a "Responses" section showing a 200 OK status. The interface also features a search bar and a sidebar with icons for various editor functions.

```
! openapi: 3.1.yml 9+ X
Users > skryukov > Desktop > ! openapi: 3.1.yml > {} components > {} schemas > {} Error > {} properties > {} message
1 openapi: 3.1.0
2
3 info:
4   title: Noah's Ark API
5   version: 0.1.0
6   description: |-
7     An API to manage and retrieve information about animals onboard Noah's
8     and check the status of the dove.
9   license:
10    name: MIT License
11    url: https://opensource.org/licenses/MIT
12   contact:
13    name: Noah
14    email: noah@ark.world
15    url: https://ark.world
16
17 tags:
18   - name: Animals
19     description: Operations concerning the management and retrieval of an
20   - name: Dove
21     description: Operations to check the status of the dove, the Ark's na
22   - name: Weather
23     description: Fetch the current weather conditions around the Ark.
24
25 servers:
26   - url: http://localhost:3000
27     description: Server for local development and testing purposes.
28
29 paths:
30   "/animals":
```

**List all animals**

Fetches a list of all animals currently onboard the Ark, with options to filter by species and paginate.

**Request**

**QUERY PARAMETERS**

species	string (animalSpecies) Species to filter animals by.
limit	integer [ 1 .. 100 ] Default: 10 Number of items to return.
page	integer >= 1 Default: 1 Page number.

**Responses**

200 OK

GET /animals >

# Linters

## ▪ Spectral

```
> spectral lint docs/openapi.yml

/noahs_ark/docs/openapi.yml
  3:6  warning  info-contact      Info object must have "contact" object
 16:5  warning  tag-description   Tag object must have "description".
 26:9  warning  operation-operationId  Operation must have "operationId".

* 3 problems (0 errors, 3 warnings, 0 infos, 0 hints)
```

*For more rules and examples see [spectral rulesets](#)*

# Mock servers

- prism

```
> prism mock docs/openapi.yml
[1:49:35 PM] > [CLI] ▶ start Prism is listening on http://127.0.0.1:4010
[1:49:47 PM] > [HTTP SERVER] get /animals i info Request received
[1:49:48 PM] > [NEGOTIATOR] i info Request contains an accept heade
[1:49:48 PM] > [VALIDATOR] ✓ success The request passed the validation
[1:49:48 PM] > [NEGOTIATOR] ✓ success Found a compatible content for *
[1:49:48 PM] > [NEGOTIATOR] ✓ success Responding with the requested st
[1:49:48 PM] > [NEGOTIATOR] i info > Responding with "200"
```

```
> curl http://127.0.0.1:4010/animals
```

```
[{"id":0,"name":"Polkan","species":"dog","sex":"male"}]
```

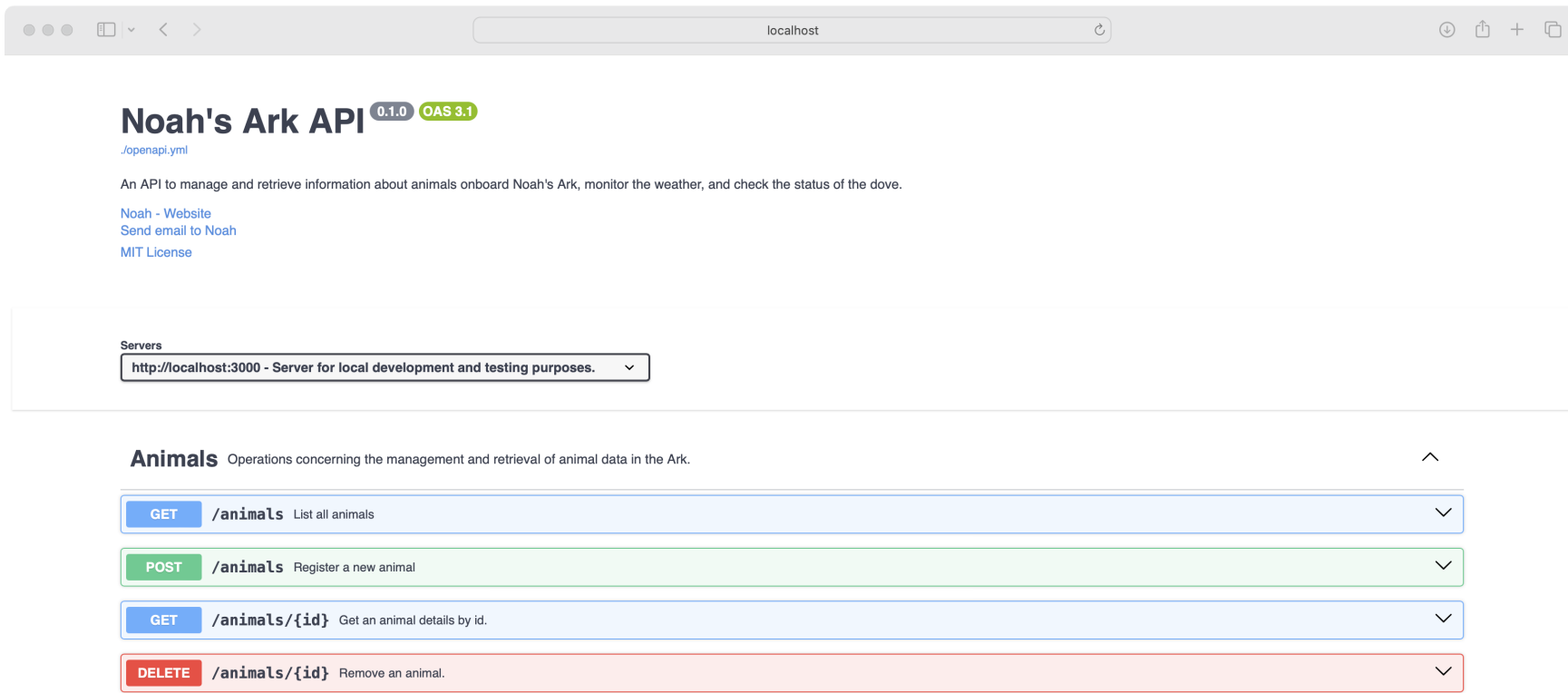
```
> curl -X POST http://127.0.0.1:4010/animals \
-H "Content-Type: application/json" \
-d '{"name":"Polkan","species":"dog","sex":"male"}'

{"id":0,"name":"Polkan","species":"dog","sex":"male"}
```



# Documentation UI

## ■ Swagger UI



**Noah's Ark API** 0.1.0 OAS 3.1

[/openapi.yml](#)

An API to manage and retrieve information about animals onboard Noah's Ark, monitor the weather, and check the status of the dove.

[Noah - Website](#)  
[Send email to Noah](#)  
[MIT License](#)

**Servers**

http://localhost:3000 - Server for local development and testing purposes.

**Animals** Operations concerning the management and retrieval of animal data in the Ark. ^

- GET** **/animals** List all animals
- POST** **/animals** Register a new animal
- GET** **/animals/{id}** Get an animal details by id.
- DELETE** **/animals/{id}** Remove an animal.

# Documentation UI

## ■ Elements

Noah's Ark API

Overview

ENDPOINTS

Animals

List all animals GET

Register a new animal POST

Get an animal details by id. GET

Remove an animal. DELETE

Feed an animal. POST

Dove >

Weather >

SCHEMAS

animalSpecies

Animal

AnimalList

Error

## List all animals

GET `http://localhost:3000/animals`

Fetches a list of all animals currently onboard the Ark, with options to filter by species and paginate.

## Request

### Query Parameters

**limit** integer  
Number of items to return.  
   
Default:

**page** integer  
Page number.  
  
Default:

**species** string  
Species to filter animals by.  
Example:

Parameters

limit : defaults to: 10

page : defaults to: 1

species : string

Send API Request

Request Sample: Shell / cURL

```
curl --request GET \
  --url http://localhost:3000/animals \
  --header 'Accept: application/json'
```

Response Example

```
1 [
2   {
3     "id": 0,
```

# Documentation UI

## ■ Redoc

The screenshot displays the Redoc API documentation interface. The left sidebar contains a search bar and a list of API endpoints under the 'Animals' category:

- GET** List all animals
- POST** Register a new animal
- GET** Get an animal details by id.
- DEL** Remove an animal.
- POST** Feed an animal.

The main content area shows the 'Animals' section with the description: 'Operations concerning the management and retrieval of animal data in the Ark.' Below this, the 'List all animals' endpoint is detailed:

**List all animals**  
Fetches a list of all animals currently onboard the Ark, with options to filter by species and paginate.

**QUERY PARAMETERS**

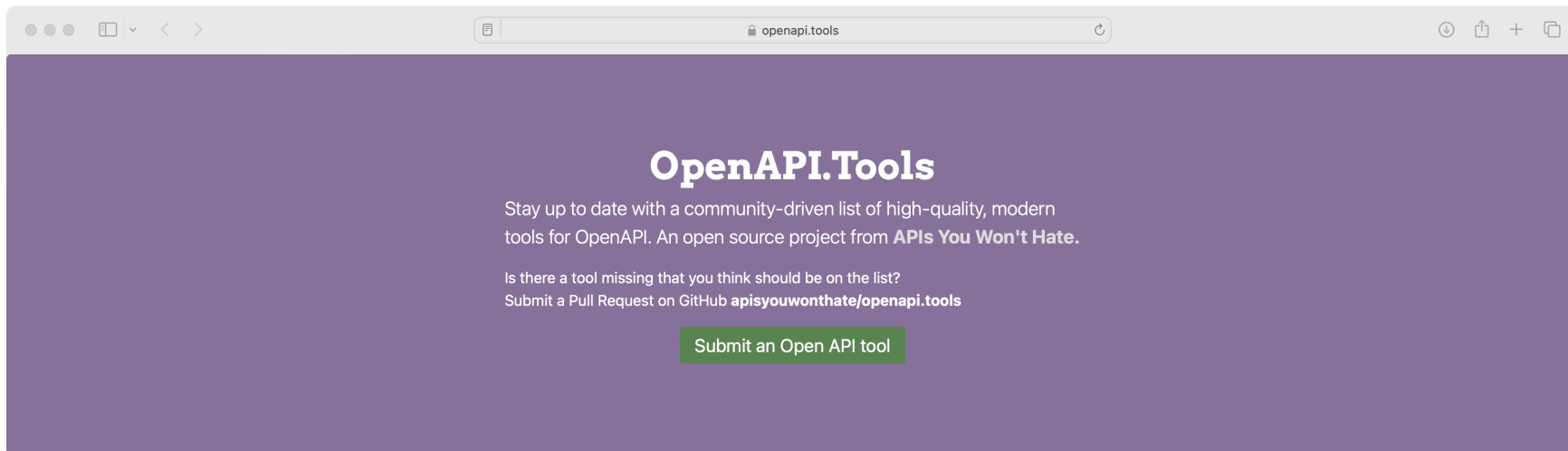
Parameter	Type	Default	Description
species	string (animalSpecies)		Examples: <code>species=dog</code> Species to filter animals by.
limit	integer [1..100]	10	Number of items to return.
page	integer >= 1	1	Page number.

The right sidebar shows the 'Response samples' section for the 'GET /animals' endpoint, displaying a 200 status code and a JSON response sample:

```
{
  "id": 0,
  "name": "Polkan",
  "species": "dog",
  "sex": "male"
}
```

At the bottom right, the page number '50' is visible.

# Many more: OpenAPI.Tools



Sponsored by [Zuplo](#) - fully-managed API gateway natively powered by OpenAPI

[Start free](#)

## Tool Types

We've organised everything into categories so you can jump to the section you're interested in.



- **Auto Generators:** Tools that will take your code and turn it into an OpenAPI Specification document
- **Converters:** Various tools to convert to and from OpenAPI and other API description formats.
- **Data Validators:** Check to see if API requests and responses are lining up with the API description.
- **Description Validators:** Check your API description to see if it is valid OpenAPI.
- **Documentation:** Render API Description as HTML (or maybe a PDF) so slightly less technical people can figure out how to work with the API.
- **DSL:** Writing YAML by hand is no fun, and maybe you don't want a GUI, so use a Domain Specific Language to write OpenAPI in your language of choice.
- **Gateways:** API Gateways and related tools that have integrated support for OpenAPI

# Collaboration

**“Awesome work, but I wanted to feed animals  
NOT the Animals Feed”**

A new documentation is ready

# Testing against specification

- write specification 
- discuss it with the team 
- add tests to conform the specification
- implement API



# Rails + Documentation First

- committee (OpenAPI 3.0)
  - battle tested gem
- openapi\_first + json\_schemer (OpenAPI 3.0, 3.1)
  - full JSON Schema 2020-12 support
- skooma + json\_skooma (OpenAPI 3.1)
  - many advantages...

# Skooma & JSON Skooma – sugar for your APIs

- full-ish OpenAPI 3.1 and JSON Schema support
- flexible and extensible design
- RSpec helpers out of the box
- cool name with a reference to The Elder Scrolls
- I'm the author

*100% unbiased opinion*

# Include Skooma into RSpec

```
# spec/rails_helper.rb

RSpec.configure do |config|
  # ...
  path_to_openapi = Rails.root.join("docs", "openapi.yml")
  config.include Skooma::RSpec[path_to_openapi], type: :request
end
```

# Validate OpenAPI document

- ``expect(skooma_openapi_schema).to be_valid_document``
  - validates OpenAPI document against the specification
  - uses JSON Schema validation under the hood

# Validate OpenAPI document

```
# spec/openapi_spec.rb

require "rails_helper"

describe "OpenAPI document", type: :request do
  subject(:schema) { skooma_openapi_schema }

  it { is_expected.to be_valid_document }
end
```

# Validate endpoints

- ``conform_request_schema``
  - **validates request path, query, headers, and body**
- ``conform_response_schema(status)``
  - **validates response status, headers, and body**
  - **testing error responses**
- ``conform_schema(status)``
  - **validates both request and response**
  - **happy path testing**

# Validate endpoints

```
# spec/requests/feed_spec.rb

require "rails_helper"

describe "/animals/:animal_id/feed" do
  let(:animal) { create(:animal, :unicorn) }

  describe "POST" do
    subject { post "/animals/#{animal.id}/feed", body:, as: :json }

    let(:body) { {food: "apple", quantity: 3} }

    it { is_expected.to conform_schema(200) }

    context "with wrong food type" do
      let(:body) { {food: "wood", quantity: 1} }

      it { is_expected.to conform_schema(422) }
    end
  end
end
```

# Red, Green, Refactor

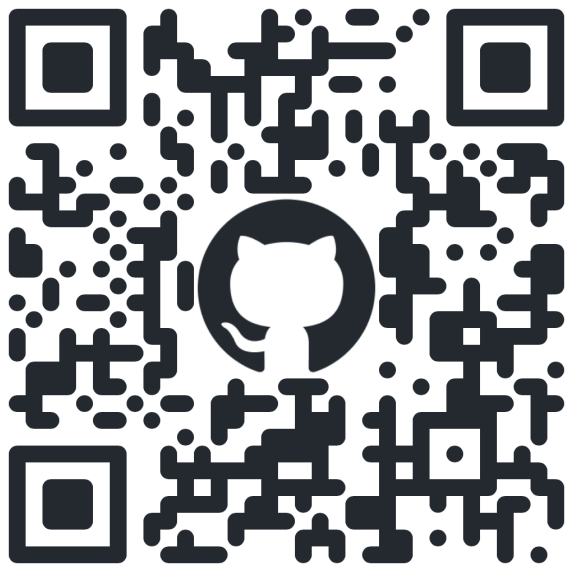
## Validation Result:

```
{"valid"=>false,  
"instanceLocation"=>"",  
"keywordLocation"=>"",  
"absoluteKeywordLocation"=>"urn:uuid:1b4b39eb-9b93-4cc1-b6ac-32a25d9bff50#",  
"errors"=>  
  [{  
    "instanceLocation"=>"",  
    "keywordLocation"=>  
      "/paths/~1animals~1{animalId}~1feed/post/responses/200"/  
      "/content/application~1json/schema/required",  
    "error"=>  
      "The object is missing required properties"/  
      " [\"animalId\", \"food\", \"amount\"]" ]}]  
# ./spec/requests/feed_spec.rb:12:in `block (3 levels) in <top (required)>'
```





# Start your path to the solid ground with Skooma



<https://github.com/skryukov/skooma>

Further along the path:

- rack middleware for input validations
- webhooks and callbacks support
- XML responses support
- i18n for error messages
- and much more...

# Tips and tricks

- begin small
  - it's ok to begin with just one endpoint
- transitioning from code/test-first approach?
  - generate OpenAPI document from your existing code and iterate on it

# Tips and tricks

- add Spectral to your CI/CD pipeline
- use custom linter rules to enforce your API design
- be inspired by Real-World Spectral Rulesets

# Tips and tricks

- read JSON Schema keywords specs to level up your validations
- for example: use ``unevaluatedProperties: false`` \* in schemas to avoid exposing internal attributes

```
type: object
unevaluatedProperties: false
properties: # ...
```

\* use ``additionalProperties: false`` for OpenAPI < 3.1

# Resources

## OpenAPI

- Getting started
- Specification

## JSON Schema

- JSON Schema in 5 minutes
- Understanding JSON Schema
- Reference documentation for JSON Schema

# Resources

## Talks

- [RailsConf 2018: Automating Empathy](#) by Ariel Caplan
- [Contract First API Design with OpenAPI V3](#) by Darrel Miller
- [Introduction to OpenAPI](#) by Lorna Jane Mitchell

# Recap

- Use Documentation First Approach:
  - enable collaboration and early feedback
- Use OpenAPI:
  - write specification in a language agnostic format
  - improve your documentation with linters
  - explore a vast ecosystem of tools
- Use Skooma:
  - test your application against the OpenAPI document
  - save time and effort



# Let there be docs

Svyatoslav Kryukov  
Evil Martians



✂ skryukov

🌀 skryukov\_dev



✂ evilmartians

🌀 evilmartians

Our blog

<https://evilmartians.com/chronicles>

Thank you 🕊️