



# OpenAPIs are everywhere

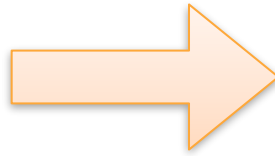
EclipseCon 2018 - Jérémie Bresson - 2018-10-23

# OpenAPI



# Swagger vs OpenAPI: it is the same!

Swagger



OpenAPI



# Jérémie Bresson

Java developer



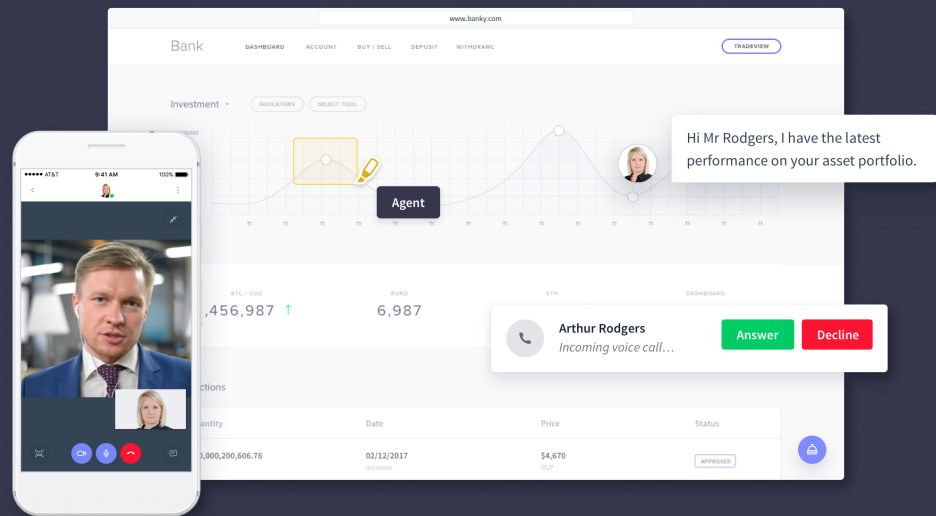
@j2r2b



jmini

Open-source contributor





# Conversational Banking made real

Switzerland | Germany | United Kingdom | Czech Republic | Bulgaria | US

[info@unblu.com](mailto:info@unblu.com)

# OpenAPIs are everywhere



# OpenAPIs are everywhere



Jens Reimann

## Apache Camel Java DSL in combination Eclipse Kura Wires

by Jens Reimann at September 19, 2018 08:30 AM

In [part #1](#) and [part #2](#), we saw how easy it is to interface [Apache Camel](#) with [Kura Wires](#). Simply by re-using some existing functionality. A few lines of XML, Groovy and you can already build an IoT solution based on the Camel ecosystem and the Eclipse Kura runtime. This part will focus on the Java DSL of Apache Camel.

It will also take into account, that when you develop and deploy an application, you need some kind of development, test and integration environment. When you build something, no matter how big, based on Camel or Kura Wires, you do want to test it. You want to have unit tests, and the capability to automatically test if your solution works, or still works after you made changes.

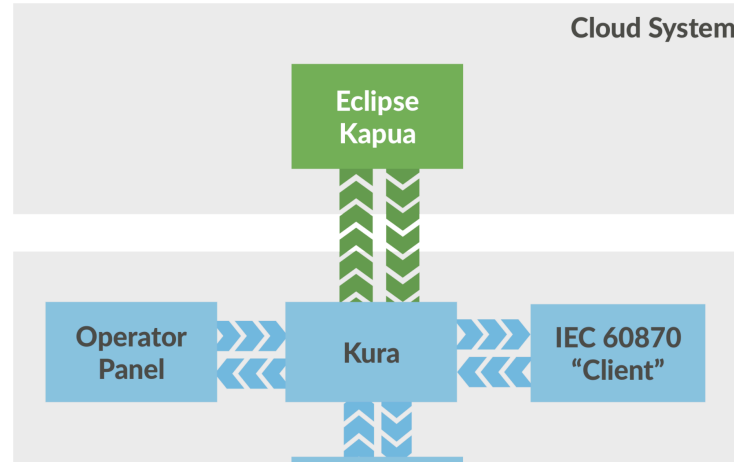
Using Kura Wires alone, this can be a problem. But Camel offers you a way to easily run your solution in a local IDE, debugging the whole process. You can have extra support for debugging Camel specific constructs like routes and endpoints. Camel has support for JUnit and e.g. using the "seda" endpoints, you can in create an abstraction layer between Camel and Wires.

### The goal

I'll make this one up (and yes, let's try to keep it realistic). We have a device, and his device allows to set two parameters for its operation (P1 and P2, both floating points). Now we already have the device connection set up in Kura. Maybe using Modbus, or something else. Kura can talk to it using Kura Wires and that is all that counts.

Now we do get two additional requirements. There is some kind of operating panel next to the device, which should allow viewing and setting those parameters locally. Also, those parameters should be accessible, using IEC 60870-5-104, for an additional device, right next to the Kura gateway.

All of those operations have to be local only, and still work when no connection to the cloud is possible. But of course, we don't want to lose the ability to monitor the parameters from our cloud system.



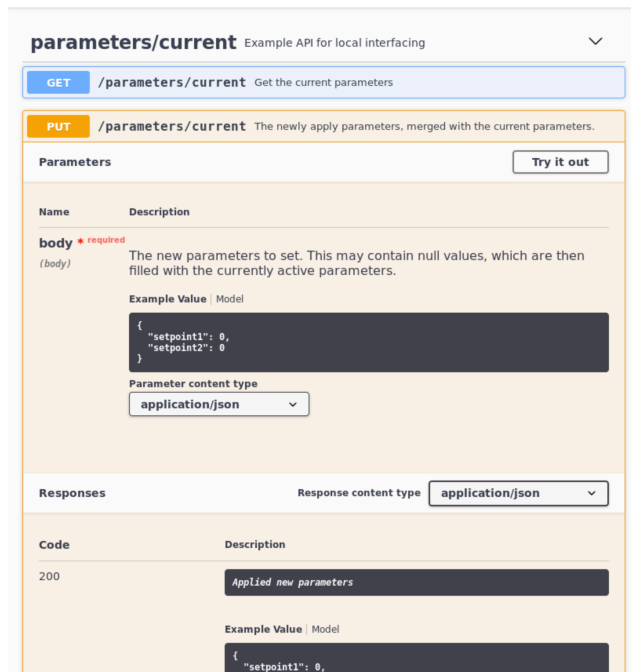
# OpenAPIs are everywhere

additional you will need to install the following dependencies:

- <https://repo1.maven.org/maven2/de/dentrassi/kura/addons/de.dentrassi.kura.addons.camel.iec60870/0.6.1/de.dentrassi.kura.addons.camel.iec60870-0.6.1.dp>
- <https://repo1.maven.org/maven2/de/dentrassi/kura/addons/de.dentrassi.kura.addons.camel.jetty/0.6.1/de.dentrassi.kura.addons.camel.jetty-0.6.1.dp>
- <https://repo1.maven.org/maven2/de/dentrassi/kura/addons/de.dentrassi.kura.addons.camel.swagger/0.6.1/de.dentrassi.kura.addons.camel.swagger-0.6.1.dp>

This will install the support for REST APIs, backed by Jetty. As Kura already contains Jetty, it only makes sense to re-use those existing components.

Once the component is deployed and started, you can navigate your web browser to <http://:8090/api>. This should bring up the Swagger UI, showing the API of the routes:



The screenshot shows the Swagger UI for the `parameters/current` API. The interface includes a header with the API title and a dropdown menu. Below the header, there are two main sections: `GET /parameters/current` and `PUT /parameters/current`. The `PUT` section is currently selected and expanded, showing a `Parameters` body with a description, an example value, and a dropdown for `Parameter content type` set to `application/json`. Below this, the `Responses` section shows a `200` response with a description and an example value.

**parameters/current** Example API for local interfacing

**GET** /parameters/current Get the current parameters

**PUT** /parameters/current The newly apply parameters, merged with the current parameters.

**Parameters** [Try it out](#)

Name	Description
<b>body</b> <span style="color:red">required</span>	The new parameters to set. This may contain null values, which are then filled with the currently active parameters.

**Example Value** | Model

```
{
  "setpoint1": 0,
  "setpoint2": 0
}
```

**Parameter content type**

`application/json`

**Responses** **Response content type** `application/json`

Code	Description
200	Applied new parameters

**Example Value** | Model

```
{
  "setpoint1": 0,
```



# OpenAPIs are everywhere

## System Dashboard

### Introduction



Welcome to unblu JIRA

New to JIRA? Check out the [JIRA User](#)

Filter by keyword ▾

- About
- Getting Started
- Authentication
- Permissions
- Expansion
- Pagination
- Ordering
- Asynchronous Methods
- Experimental Methods
- Special Headers
- Error responses

### About

This is the reference for the Jira Cloud REST API. This API is the primary way to interact with Jira re notel building an app, scripting interactions with Jira or developing any other integration. This page document available in Jira Cloud, along with expected HTTP response codes and sample requests.

Looking for the REST API reference for Jira Server? Follow the [Jira Server REST API](#) link.

#### A note about the V3 API

The v3 API is currently in beta. Note that while all endpoints from the v2 API are available, they are currently under development. This means that any endpoint can change at any time, although we will not introduce breaking changes without advanced notice.

### Getting Started

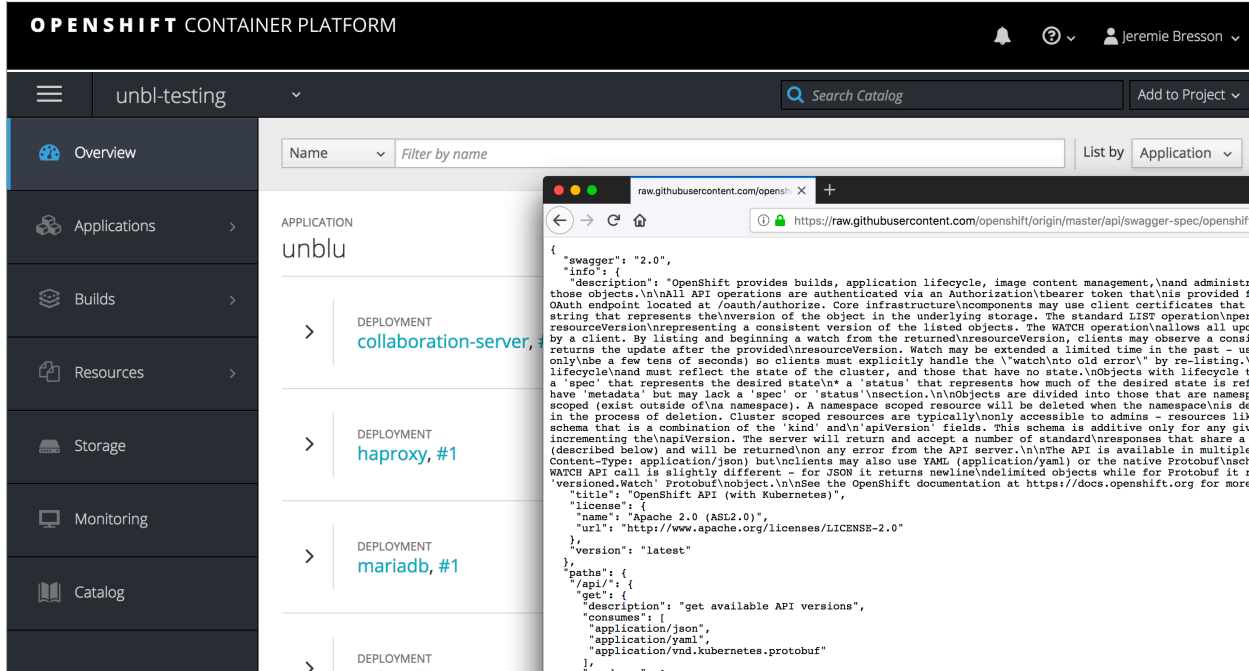
If you haven't integrated with Jira Cloud before, start with [Integrating with Jira Cloud](#) guide. The guide will introduce you to the Atlassian Connect framework, as well as Jira features and services that you can use when building an app. Then, read our [Getting started](#) guide to learn how to set up a development environment and build a Jira Cloud app.

### Authentication

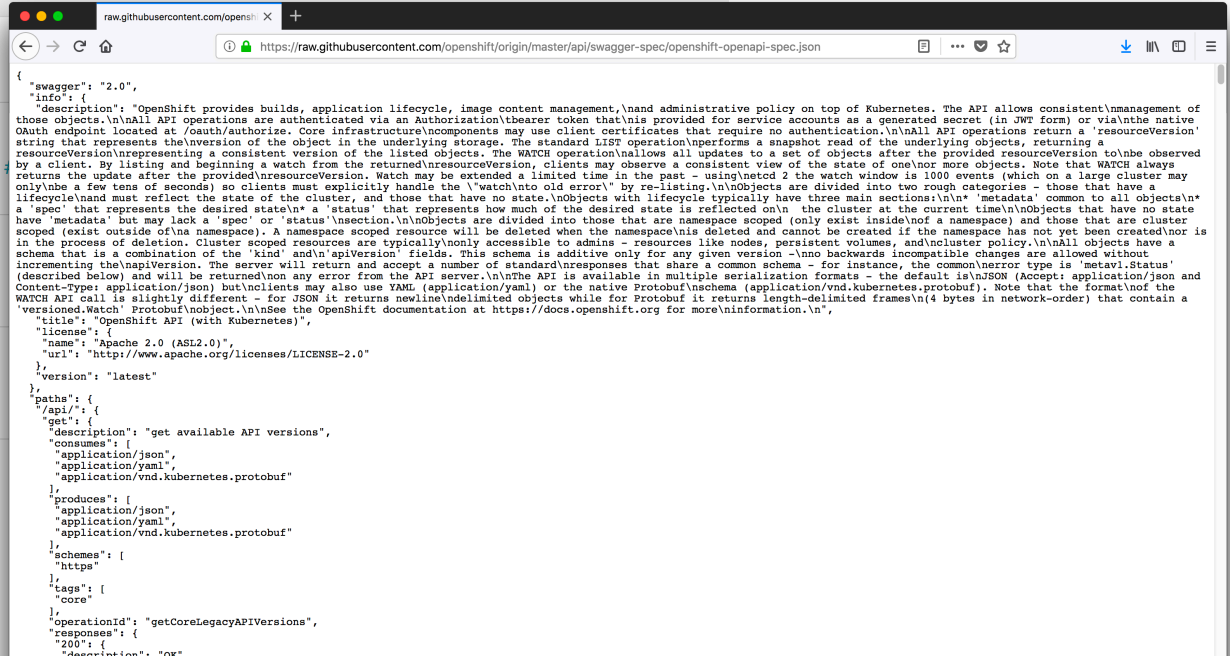
Run in Postman

Download OpenAPI spec

# OpenAPIs are everywhere



The screenshot shows the OpenShift Container Platform interface. At the top, it says "OPENSIFT CONTAINER PLATFORM" and "unbl-testing". Below that, there's a search bar and "Add to Project". The left sidebar has navigation options: Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main content area shows the "unblu" application page with a list of deployments: "collaboration-server", "haproxy, #1", and "mariadb, #1".



The screenshot shows a browser window displaying the OpenAPI specification for the unblu API. The URL is <https://raw.githubusercontent.com/openshift/origin/master/api/swagger-spec/openshift-openapi-spec.json>. The JSON content is as follows:

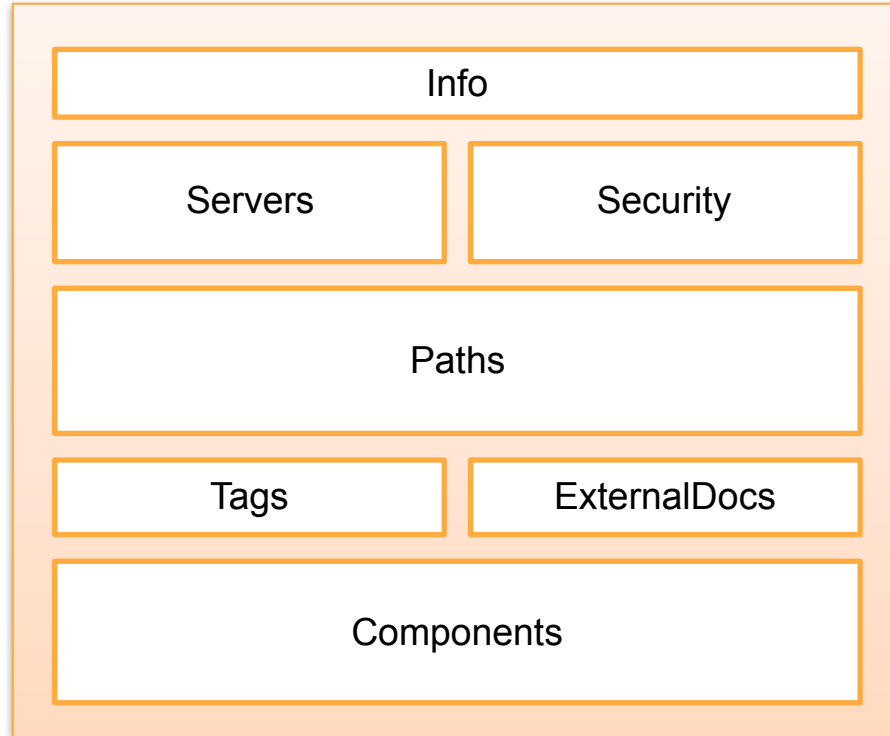
```
{
  "swagger": "2.0",
  "info": {
    "description": "OpenShift provides builds, application lifecycle, image content management, and administrative policy on top of Kubernetes. The API allows consistent management of these objects. All API operations are authenticated via an Authorization bearer token that is provided for service accounts as a generated secret (in JWT form) or via the native OAuth endpoint located at /oauth/authorize. Core infrastructure components may use client certificates that require no authentication. All API operations return a 'resourceVersion' string that represents the version of the object in the underlying storage. The standard LIST operation performs a snapshot read of the underlying objects, returning a resourceVersion representing a consistent version of the listed objects. The WATCH operation allows all updates to a set of objects after the provided resourceVersion to be observed by a client. By listing and beginning a watch from the returned resourceVersion, clients may observe a consistent view of the state of one or more objects. Note that WATCH always returns the update after the provided resourceVersion. Watch may be extended a limited time in the past - using the watch window is 1000 events (which on a large cluster may only be a few tens of seconds) so clients must explicitly handle the 'watch too old error' by re-listing. Objects are divided into two rough categories - those that have a lifecycle and must reflect the state of the cluster, and those that have no state. Objects with lifecycle typically have three main sections: 'metadata' common to all objects, a 'spec' that represents the desired state, and a 'status' that represents how much of the desired state is reflected on the cluster at the current time. Objects that have no state have 'metadata' but may lack a 'spec' or 'status' section. Objects are divided into those that are namespace scoped (only exist inside of a namespace) and those that are cluster scoped (exist outside of a namespace). A namespace scoped resource will be deleted when the namespace is deleted and cannot be created if the namespace has not yet been created or is in the process of deletion. Cluster scoped resources are typically only accessible to admins - resources like nodes, persistent volumes, and cluster policy. All objects have a schema that is a combination of the 'kind' and 'apiVersion' fields. This schema is additive only for any given version - no backwards incompatible changes are allowed without incrementing the apiVersion. The server will return and accept a number of standard responses that share a common schema - for instance, the common error type is 'metav1.Status' (described below) and will be returned upon any error from the API server. The API is available in multiple serialization formats - the default is JSON (Accept: application/json and Content-Type: application/json) but clients may also use YAML (application/yaml) or the native Protobuf schema (application/vnd.kubernetes.protobuf). Note that the format of the WATCH API call is slightly different - for JSON it returns newline-delimited objects while for Protobuf it returns length-delimited frames (4 bytes in network-order) that contain a 'versioned.watch' Protobuf object. See the OpenShift documentation at https://docs.openshift.org for more information.",
    "title": "OpenShift API (with Kubernetes)",
    "license": {
      "name": "Apache 2.0 (ASL2.0)",
      "url": "http://www.apache.org/licenses/LICENSE-2.0"
    },
    "version": "latest"
  },
  "paths": {
    "/api/": {
      "get": {
        "description": "get available API versions",
        "consumes": [
          "application/json",
          "application/yaml",
          "application/vnd.kubernetes.protobuf"
        ],
        "produces": [
          "application/json",
          "application/yaml",
          "application/vnd.kubernetes.protobuf"
        ],
        "schemes": [
          "https"
        ]
      },
      "core": {
        "operationId": "getCoreLegacyAPIVersions",
        "responses": {
          "200": {
            "content": "or"
```

# Content of an OpenAPI Specification



# Content of an OpenAPI Specification

OpenAPI v3



JSON  
or  
YAML

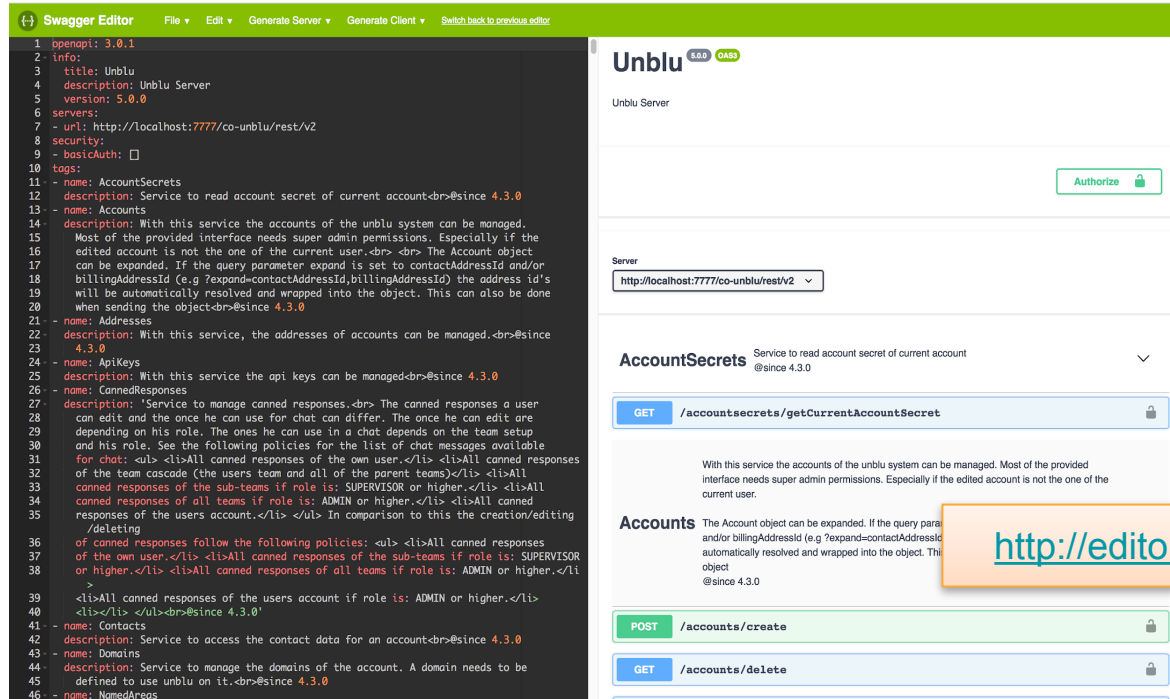
# Content of an OpenAPI Specification

```
1  openapi: 3.0.1
2  info:
3    title: A TODO-Task list application
4    description: A simple application to handle tasks.
5    version: 1.0.0
6    license:
7      name: Apache-2.0
8      url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
9  servers:
10 - url: 'http://localhost:8080/todoapp'
11 tags:
12 - name: task
13   description: Task management
14 paths:
15   /Task:
16     get:
17       operationId: getAllTasks
18       description: Get the list of all tasks
19       tags:
20       - task
21       responses:
22         200:
23           description: a list of all tasks
```

# Editors for OpenAPI specifications



# Swagger Online Editor



The image shows the Swagger Editor interface. On the left, a code editor displays a Swagger specification for an API. The specification includes metadata (title: Unblu, version: 5.0.0), a server URL, and several endpoints. The endpoints are:

- AccountSecrets**: Service to read account secret of current account. Endpoint: `/accountsecrets/getCurrentAccountSecret` (GET).
- Accounts**: The Account object can be expanded. Endpoints: `/accounts/create` (POST) and `/accounts/delete` (GET).

On the right, the rendered API documentation is shown. It features the Unblu logo, a server selection dropdown (set to `http://localhost:7777/co-unblu/rest/v2`), and a list of endpoints. The **AccountSecrets** endpoint is expanded, showing its description: "With this service the accounts of the unblu system can be managed. Most of the provided interface needs super admin permissions. Especially if the edited account is not the one of the current user." An orange callout box highlights the URL `http://editor.swagger.io/`.

# Eclipse IDE plugin: KaiZen-OpenAPI-Editor



```
unblu-openapi.yaml
1 openapi: 3.0.1
2 info:
3   title: Unblu
4   description: Unblu Server
5   version: 5.0.0
6 servers:
7   - url: http://localhost:7777/unblu/rest/v1
8 security:
9   - basicAuth: []
10 tags:
11   - name: AccountSecrets
12     description: Service to read account secret of current account<br>@since 4.3.0
13   - name: Accounts
14     description: With this service the accounts of the unblu system can be managed.
15       Most of the provided interface needs super admin permissions. Especially if the
16       edited account is not the one of the current user.<br> <br> The Account object
17       can be expanded. If the query parameter expand is set to contactAddressId and/or
18       billingAddressId (e.g ?expand=contactAddressId,billingAddressId) the address id's
19       will be automatically resolved and wrapped into the object. This can also be done
20       when sending the object<br>@since 4.3.0
21   - name: Addresses
22     description: With this service, the addresses of accounts can be managed.<br>@since
23     4.3.0
24   - name: ApiKeys
25     description: With this service the api keys
26   - name: CannedResponses
27     description: 'Service to manage canned resp
28     can edit and the once he can use for chat
29     depending on his role. The ones he can use in a chat depends on the team setup
30     and his role. See the following policies for the list of chat messages available
31     for chat: <ul> <li>All canned responses of the own user.</li> <li>All canned responses
32     of the team cascade (the users team and all of the parent teams)</li> <li>All
33     canned responses of the sub teams if role is: SUPERVISOR or higher.</li> <li>All'
```

<https://github.com/RepreZen/KaiZen-OpenAPI-Editor>



# Postman client

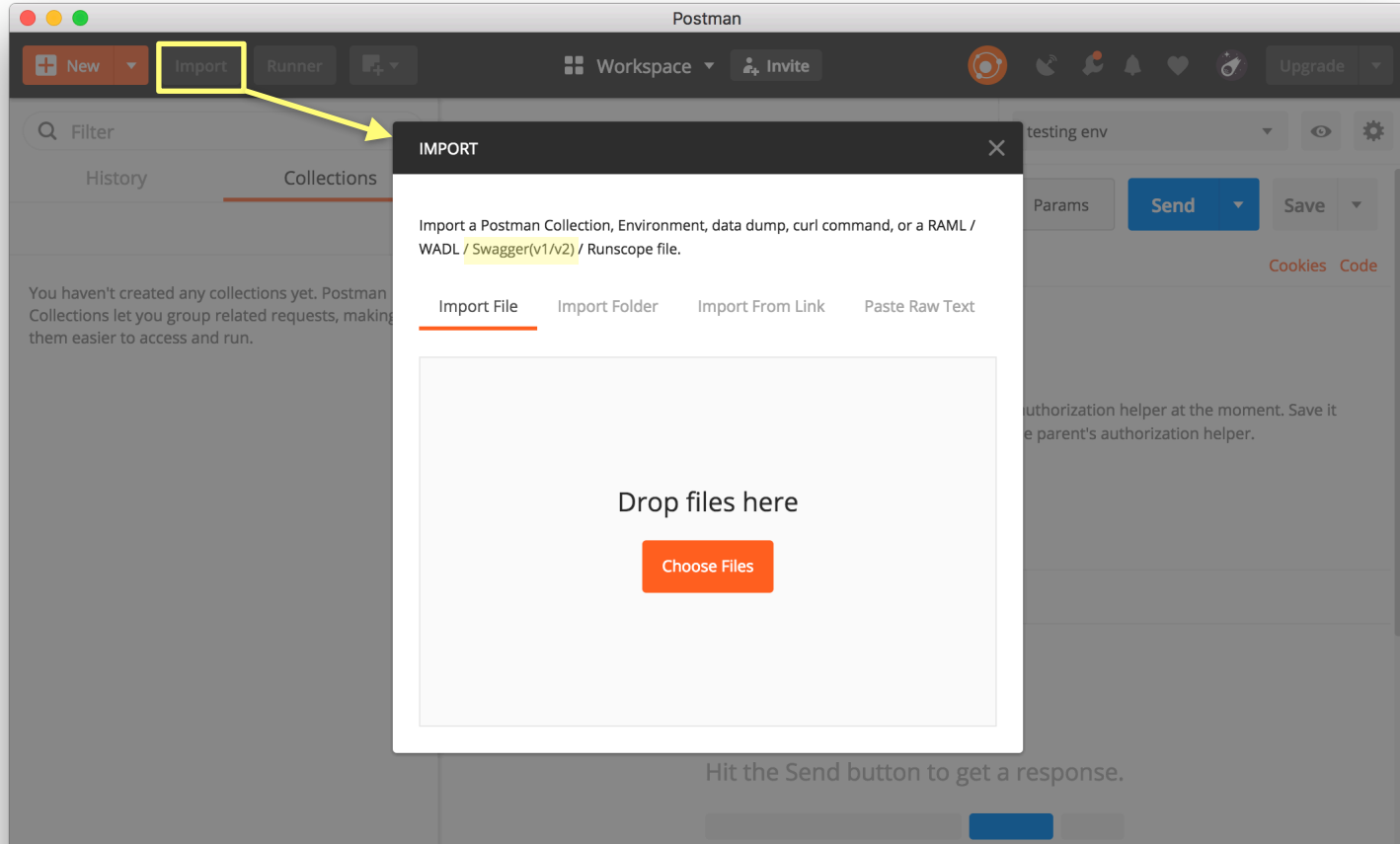


# Postman



The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named "A TODO-Task list application" containing four requests: "getAllTasks" (GET), "createTask" (POST), "updateTask" (PUT), and "deleteTask" (DEL). The main workspace is configured for a GET request to "http://localhost:8080/todoapp/task". The "Authorization" tab is active, showing a dropdown menu set to "Inherit auth from pa...". A message states: "This request is using an authorization helper from A TODO-Task list application". A blue "Send" button is visible. Below the configuration, a "Response" section is present. An orange callout box at the bottom right contains the URL <https://www.getpostman.com/>. At the very bottom, text reads "Hit the Send button to get a response."

# Postman: import OpenAPI (v2)



# Server-side



# Support for OpenAPI (Java Server)



# Adding OpenAPI to a JAX-RS project



**Thorntail**  
(a.k.a WildFly Swarm)  
<https://thorntail.io/>



<https://openliberty.io/>

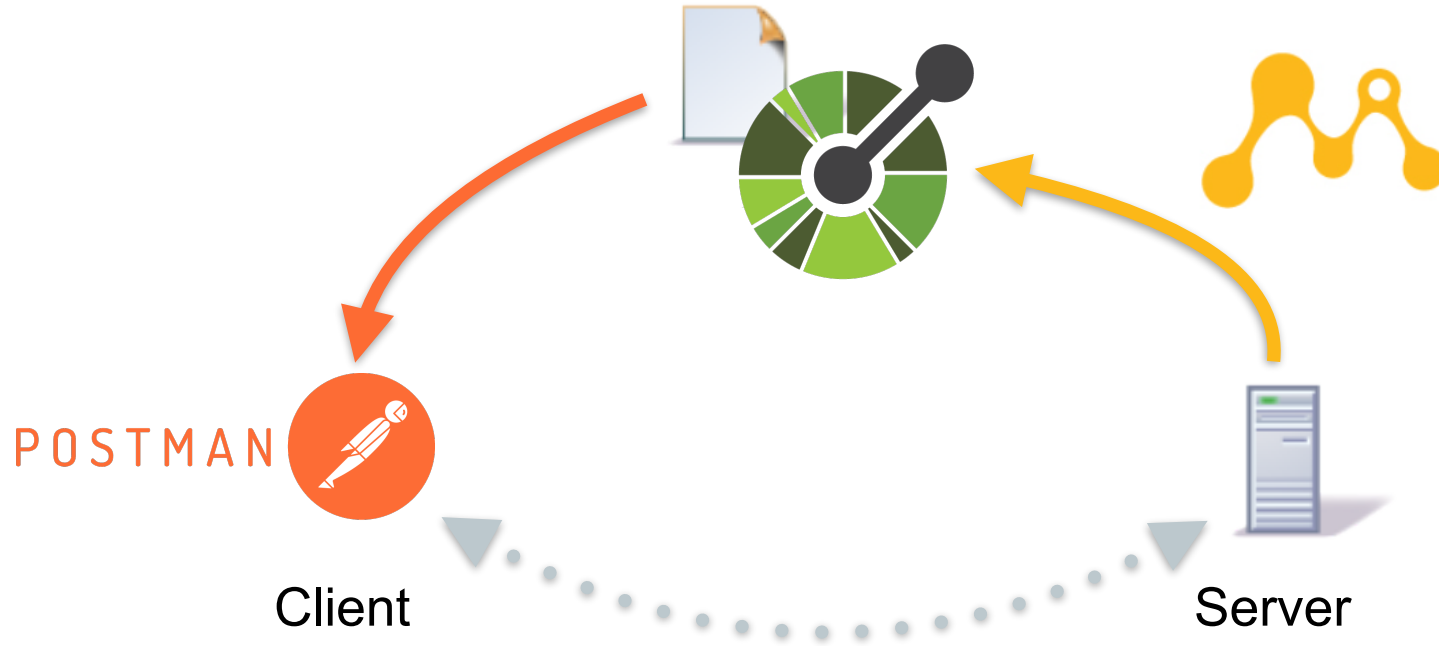
# Adding OpenAPI to a JAX-RS project

- Add the additional maven dependency:

```
<dependency>  
  <groupId>io.thorntail</groupId>  
  <artifactId>microprofile-openapi</artifactId>  
</dependency>
```

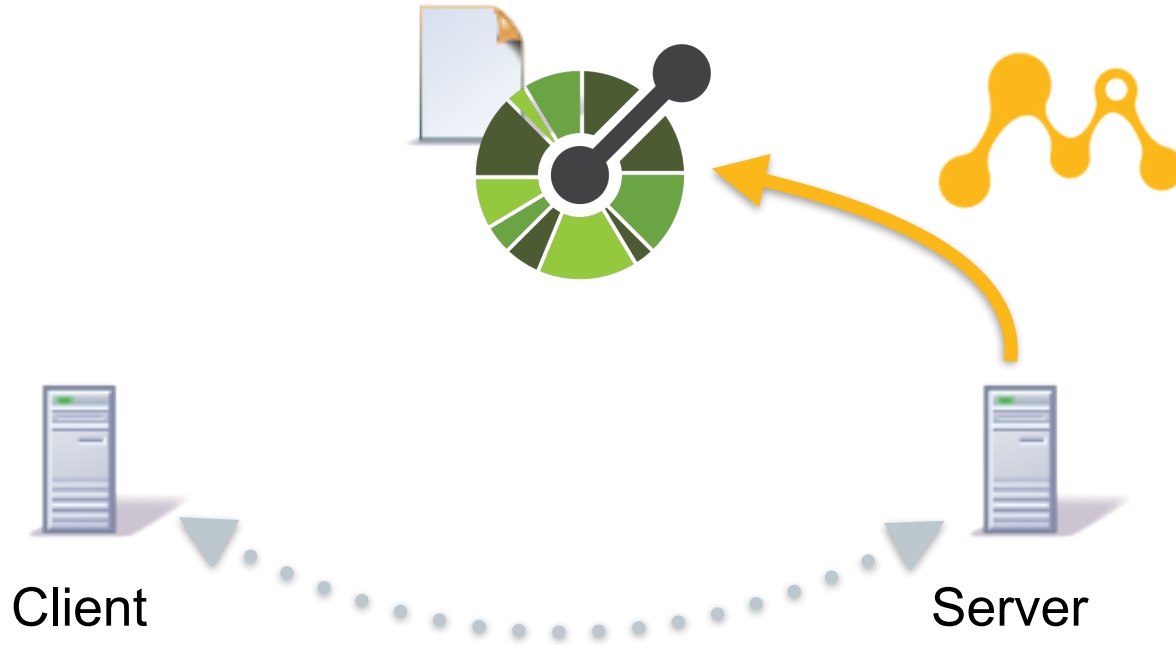
- Add OpenAPI annotations (next to the JAX-RS ones)

# Code first approach





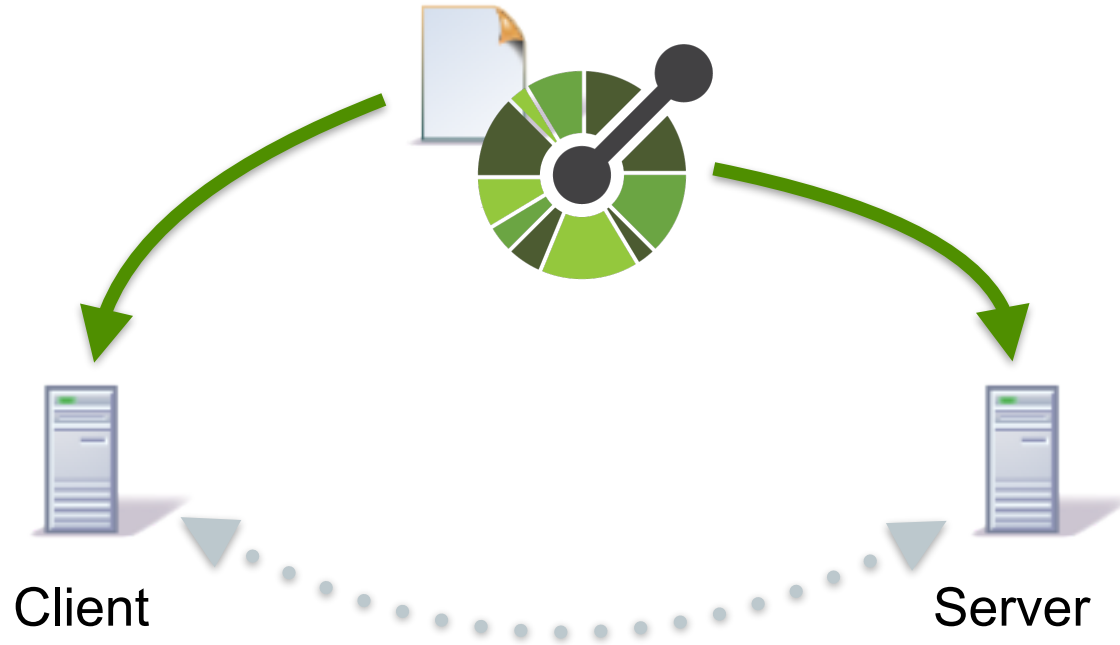
# Code first approach



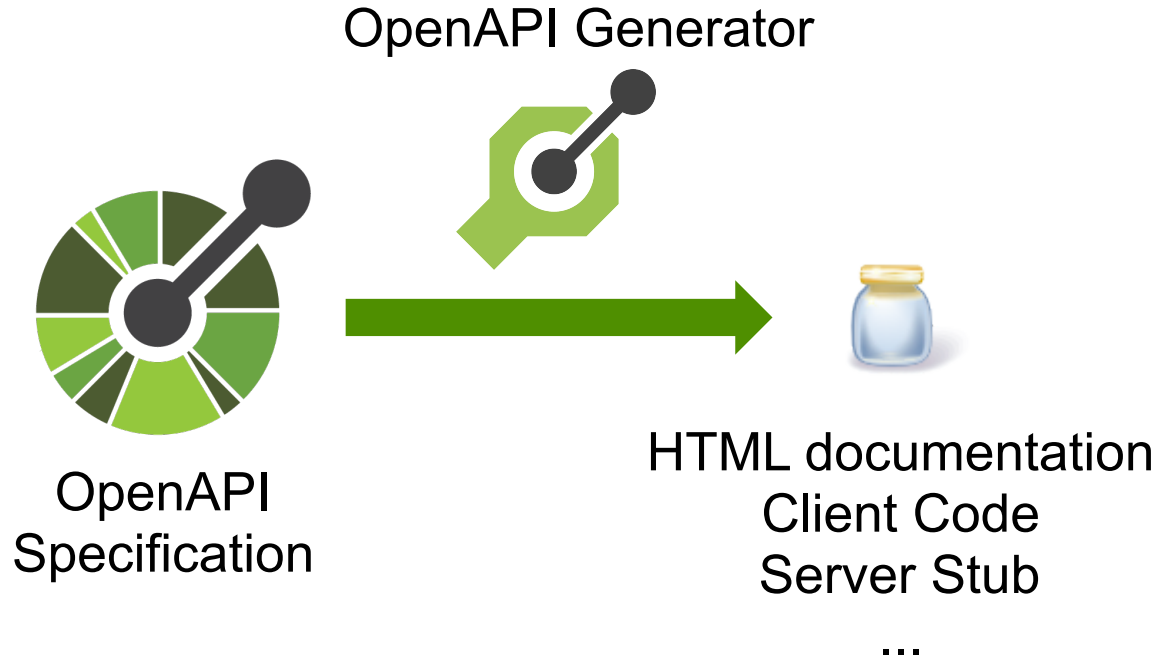
# Specification first approach



# Start with a Specification



# Code generator: OpenAPI-Generator





# OpenAPI-Generator - Usage

Generate a java client for the example application

```
java -jar ./openapi-generator-cli-3.3.1.jar generate
    -i ../OpenAPI-Spec/todoapp.yaml
    -g java
    -o out/
```

# OpenAPI: an ecosystem



# A lot of tools are available for OpenAPI

- Editors
- Tool to generate documentation
- Breaking change detection
- Integration with existing tools (Postman, ...)
- Code generation  
(OpenAPI-Generator, AutoRest, Swagger-Codegen, ... )

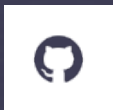




Thank you for  
your attention



@j2r2b



jmini

Code Examples:

<https://github.com/jmini/ece2018-openapi>



## Evaluate the Sessions

Sign in and vote at [eclipsecon.org](https://eclipsecon.org)

-1

0

+1

