OAS2Tree: Visual API-First Design

Souhaila Serbout and Cesare Pautasso

Software Institute (USI), Lugano, Switzerland
first-name.last-name@usi.ch

Abstract. OAS2tree is a tool designed to transform OpenAPI Specification (OAS) documents into tree-like visualizations, aiding in the understanding and navigation of the structure of REST APIs. By converting the detailed, verbose, and often complex OAS files into a visual tree structure, OAS2tree simplifies the comprehension of a Web API, highlighting the hierarchical relationships between endpoints, operations, and parameters. This visual representation is particularly useful for developers and stakeholders who need a quick overview of an API without delving into the intricate details of its technical specifications. OAS2tree can be integrated into the IDE through a Visual Studio code extension or used as a standalone web application. The tool currently has about 400 users and has been used on teaching, research, and development projects. In this paper, we present the design and implementation of OAS2tree, highlighting its features and use cases. We also highlight the limitations of the current version and discuss future improvements and potential extensions.
Demo Video Link: https://youtu.be/E48c9Rwntz8

1 Introduction

Web APIs are essential for enabling seamless communication between software systems [10]. However, their sheer size and complexity [7] can make them challenging to understand [4]. While visualizing the overall architecture can significantly aid in understanding the interactions between a system's components, focused visualizations of crucial elements within a complex system, such as web APIs, can clarify the structure, help ensure the correct flow of data, and determine which operations should be exposed.

OAS2Tree dynamically generates visual representations of APIs described using the OpenAPI Specification (OAS) [5]. The tree-like visualization represents the API endpoints, request/response elements, and parameters. By offering developers a visual representation instantaneously synchronized from the OpenAPI description text, the tool facilitates obtaining valuable insights and a deeper understanding of the API structure. Furthermore, our tool goes beyond basic visualization capabilities by proactively identifying and highlighting design smells—common issues or inefficiencies in API design. These design smells have been extensively discussed in our previous research study on API structural patterns and design flaws [9].

In this paper, we outline the main features of OAS2tree. By offering developers the ability to interact with visual representations and detect design smells early on, the tool empowers developers to ensure they create the wanted API and consumers to inspect and select their chosen API. By providing real-time visual representations of API endpoint and operation structures, the tool facilitates validating the consistent and regular design of API endpoints as a tree of URL paths with color-coded HTTP methods as tree leaves. Additionally, its integration of design smell detection capabilities assists developers in identifying potential issues and refining their APIs. Ultimately, our visualization tool aims to enhance the collaboration and efficiency of incremental API design and review processes, resulting in the creation of well-designed APIs.

The tool is available as both a standalone web app [1] and as a Visual Studio Code (VSCode) extension [2], catering to the diverse needs of developers. The web app version is designed for those who want a quick and convenient way to visualize an API without the need for local tool installation and setup. On the other hand, the VSCode extension version is intended for developers who prefer to have the visualization tool seamlessly integrated into their development environment.

2 From OpenAPI to API Tree

To visualize the OpenAPI specification as a tree structure, we transform the flat list of paths extracted from an API description into a hierarchical structure by breaking down the paths into segments. When a segment is shared between different paths, we check if these segments share the same sequence of parent segments. If they do, the segments are merged into a single node, as they refer to the same container resource. Once the path segments tree structure is complete we attach to each node the set of HTTP methods of the corresponding endpoint.

OpenAPI also includes details related to the responses of each endpoint and the parameters that can be passed to the endpoint. We attach these details to the corresponding HTTP method node. In the current version of OAS2Tree, we only visualize the response status codes but do not further drill down to show the data models of the request or response schemas.

Considering the example in Listing 1.1 of API paths defined in an OpenAPI document:

Listing 1.1: Example of API paths description ins an OpenAPI document

```
paths:
  /users:
    get:
      summary: "Retrieve the list  of all registered users."
      responses:
        "200":
          description: "A list of users."
  /users/{id}/details:
    get:
```

```
    summary: "Retrieve details of a user by their ID."
    responses:
      "200":
        description: "The user information."
      "404":
        description: "User not found."
/posts:
  get:
    summary: "List all posts"
    responses:
      "200":
        description: "A list of all available posts."
        parameters:
        - name: limit
          in: query
          description: "The number of posts to return."
          type: integer
        - name: offset
          in: query
          description: "The number of posts to skip."
          type: integer
  post:
    summary: "Create a new post."
    responses:
      "201":
        description: "The created post."
```

The paths `/users` and `/users/{id}/details` share the segment `/users`, and the paths `/users/{id}/details` and `/posts` share the segment `/users/{id}`. Since these segments share the same sequence of parent segments, they are merged into one node in the tree structure. The resulting tree structure can be visualized as in Figure 1.

While the YAML or JSON syntax adopted by OpenAPI can make it hard to locate the operations that are applicable over the same resources but with

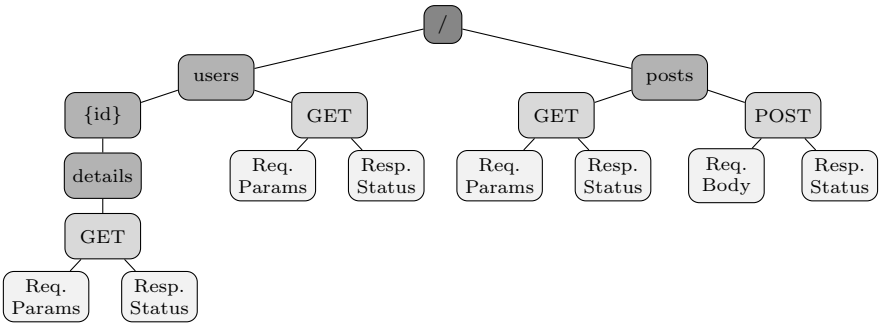


Fig. 1: Extracted API tree structure

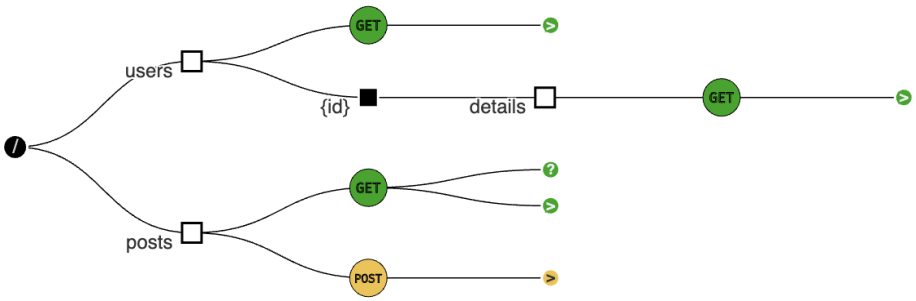


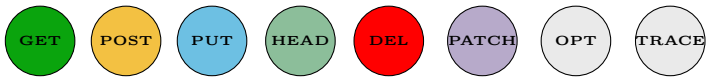
Fig. 2: OAS2Tree visualization of API in Listing 1.1

different HTTP methods, in case of large APIs with many paths, the tree structure makes it easy to see that – for example – the `/users/{id}/details` path has only a GET operation and the `/posts` path has both a GET and a POST operation. This hierarchical representation clearly shows the shared segments and their relationships within the API structure.

3 OAS2Tree Elements Graphical Notation

We defined the following set of notations to represent the different elements of the API tree structure (Figure 2).

HTTP Methods: They are visualized in a circular shape. We attribute a specific color to each HTTP method to make it easier to identify them in the visual representation. The colors we adopted are closely similar to the color coding used in Postman [6]. The colors are as follows:



Path segments: we distinguish the position of fixed URL segments (represented with a white tree node □) from the position of in-path parameters (represented with a black tree node ■). We also use a different notation (■) to distinguish unusual path segments. For example in `/api/v1/users/{userId}:posts` the last suffix of the path "`:posts`" is prefixed by a colon character as opposed to the usual forward slash.

Query parameters: we represent query parameters as a subtree of the operation node. The subtree is collapsed by default, to highlight the presence of query parameters, and the user can expand it to see which parameters are expected by the operation. The root of the subtree is represented by a question mark icon ⓘ . Then each parameter is represented by a node with the parameter name (Figure 3).

Responses: we represent the responses of an operation as a subtree of the operation node. The sub-tree is collapsed by default indicating the presence of

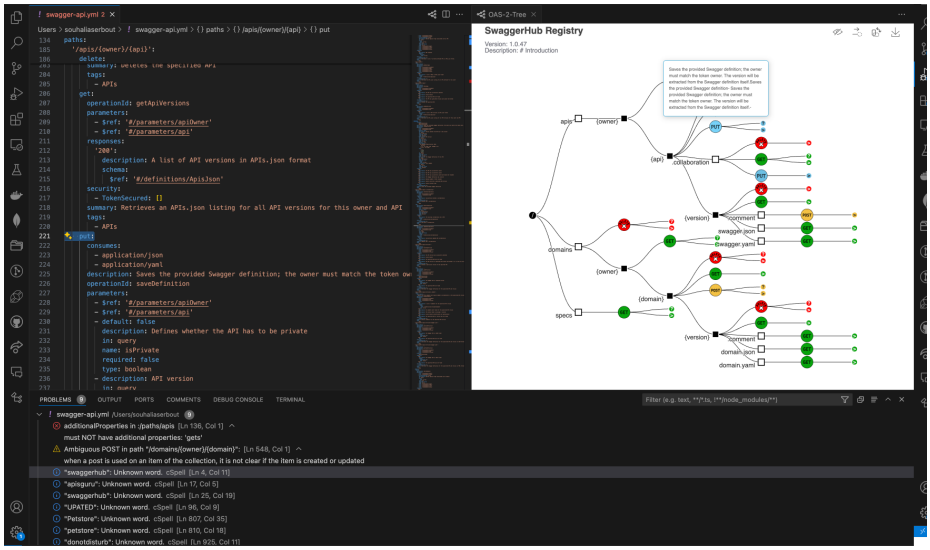


Fig. 3: Navigation between the diagram and the OAS code

responses. The user can expand it to see the details of the responses (Figure 3). The root of the subtree is represented by a greater than icon ⓥ . Each response is represented by a node with the response status code. The response code is colored in green if it is a success code (2xx), in orange if it is a client-side error (4xx), and in red if it is a server-side error (5xx).

4 OAS2Tree Features

4.1 API Spec validation Design Smells Detection

OAS2Tree can be used to detect smells in the API design described in the specification. The tool currently supports the detection design smells we empirically identified in our previous research study on API collection resource patterns [9]. The goal is to alert developers to potential design issues. The design smells detected by OAS2Tree include:

- **Ambiguous PUT and POST endpoints:** When an API contains both PUT and POST operations with similar paths, it can lead to confusion.
- **Create without delete:** When an API allows the creation of resources without providing a corresponding delete operation, it can result in data inconsistencies.
- **Delete without create:** When an API allows the deletion of resources without providing a corresponding create operation, it can lead to data loss.
- **Write-only endpoints:** When an API contains endpoints that only allow write operations without providing read capabilities, it can limit the usability of the API.

In addition, OAS2Tree validates the API specification against the OpenAPI Specification schema to ensure that it adheres to the standard, and highlights any errors or inconsistencies in the document and in the problems view as depicted in Figure 3. For both smells and validation errors, the user can navigate to the problematic element in the OAS document by clicking on the error message.

4.2 Navigation of API description through the tree visualization

The tree structure visualization allows users to navigate through the API description easily. By expanding and collapsing nodes, users can explore the API structure and view the details of each endpoint, operation, and parameter. This interactive feature provides a comprehensive overview of the API architecture, enabling users to quickly locate specific endpoints and understand their functionalities. On mouse over, the editor highlights the corresponding element in the OpenAPI document in yellow. The description of the element is displayed in the tooltip. It is also possible to navigate from the problems view to the problematic element in the OpenAPI document by clicking on the error message.

4.3 Web version of OAS2Tree

OAS2Tree is also available as a standalone web application, allowing users to visualize API structures without the need for a development environment. The web app version provides the same features as the VSCode extension, including the ability to save the current specification being visualized and share it through a unique URL with other users. The web app version is particularly useful for users who want to quickly visualize or sketch an API design from their Web browser (Figure 4). An additional feature that the web version offers is the ability to navigate a collection of API specification examples, and visualize them in the tree structure format. This feature can be useful for users who want to explore different API designs and understand the common patterns and structures used in API development, and how some features are documented in OpenAPI by other API designers[1].

[1] `http://api-ace.inf.usi.ch/openapi-to-tree/navigate-apis?limit=50&page=13`

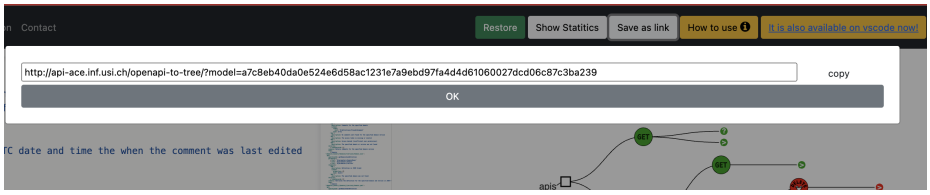


Fig. 4: Save as URL functionality in OAS2Tree Web App

5 Usage scenarios

OAS2Tree can be used by both API providers and API consumers in various use-case scenarios across the API development lifecycle.

As an API Designer & Developer:

– API Design and Documentation: During the initial stages of API design, OAS2Tree can be employed to visualize and refine the API structure. Since the visualization can also be generated from not fully complete OAS, it can serve as a tool for sketching partial API structures, and have an early overview of it.

– API Review and Design Validation: Designers can use OAS2Tree not only to ensure conformance to the OpenAPI standard, but also to ensure consistency in endpoint naming, parameter usage, and selection of HTTP methods.

– API Evolution: The tool can help find the adequate place of an extension.

As an API Consumer:

– Functionalities exploration: Client developers can use OAS2Tree to explore the endpoint structure and assess whether the API meets their requirements.

– Documentation navigation: OAS2Tree can be used to navigate the API documentation and quickly locate specific endpoints and operations, by hovering over the tree nodes, the corresponding element in the OpenAPI document is highlighted.

– Visual comparison: OAS2Tree can be used to visually compare the structure of different versions of an API, or the structures of different APIs.

6 Related Work

Most of the currently widely used API lifecycle management tools in the market[2][3][4], including paid ones, offer a Postman-like interface in their 'API Design environment', where the user can interact with the API, and test it. However, none of them offers a tree-like visualization of the API structure, that can be used to understand the overall API structure and navigate the documentation.

OAIE Sketcher[5] offers another way to visualize the endpoints by emphasizing the relationships between the schemas used in the request and response bodies. However, it lacks the hierarchical structure of the API paths and the HTTP methods, and the visualization is not kept synchronized as the corresponding textual specification changes.

The tool OpenAPItoUML [3] generates UML models from OpenAPI definitions, providing a means to visualize both API endpoint structures and API data models using class diagrams. However, it does not provide a tree-like visualization of the API paths and operations, and it does not support the detection of any design smells in the API specification or data schema issues.

OAS2Tree differentiates itself from existing tools [8] by focusing specifically on rendering OpenAPI descriptions as tree-like visualizations.

[2] https://xapihub.io/features/designAndDev
[3] https://stoplight.io/drive-api-results
[4] https://apigit.com/why-apigit/api-design
[5] https://raw.githack.com/OAIE/oaie-sketch/master/sketch.html

7 Conclusion and Future Work

OAS2Tree is a REST API visualization tool, available as a Visual Studio Code extension and a web application, that transforms OpenAPI Specifications into a simple visual tree representation. It supports OAS v2.0, v3.0, and v3.1. It provides a side panel in the VS Code editor to display the API structure as a tree. The tool can be employed by API designers to visualize and refine the API structure during the initial design phase, ensuring consistency in endpoint naming, parameter usage, and HTTP method selection. It can also be used to validate the API design and detect potential design smells early on. API consumers can use OAS2Tree to explore the functionalities of an API and understand whether it meets their requirements. In addition, it contains a navigation feature that can help to quickly locate the textual description of specific endpoints and operations.

In future work, we plan to extend the tool to support detecting additional design smells and enhance the representation of the detected issues on the tree visualization to ease their location in the overall API structure. We are also experimenting with embedding the visualization as part of the API documentation generated from the OpenAPI description.

References

1. OAS2Tree, `http://api-ace.inf.usi.ch/openapi-to-tree/`
2. OAS2Tree, `https://marketplace.visualstudio.com/items?itemName=oas2tree.oas2tree`
3. Ed-Douibi, H., Cánovas Izquierdo, J.L., Cabot, J.: OpenAPItoUML: a tool to generate UML models from OpenAPIdefinitions. In: International Conference on Web Engineering. pp. 487–491. Springer (2018)
4. Grent, H., Akimov, A., Aniche, M.: Automatically identifying parameter constraints in complex web APIs: a case study at adyen. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). pp. 71–80. IEEE (2021)
5. OpenAPI Initiative: OpenAPI specification (2021), `https://spec.openapis.org/oas/v3.1.0`
6. Postman: Postman (2021), `https://www.postman.com/`
7. Serbout, S., Di Lauro, F., Pautasso, C.: Web apis structures and data models analysis. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). pp. 84–91. IEEE (2022)
8. Serbout, S., Hurtado, D.C.M., Pautasso, C.: Interactively exploring api changes and versioning consistency. In: 11th IEEE Working Conference on Software Visualization (VISSOFT 2023). pp. 28–39. IEEE, IEEE, Bogota, Colombia (October 2023)
9. Serbout, S., Pautasso, C., Zdun, U., Zimmermann, O.: From OpenAPI fragments to api pattern primitives and design smells. In: 26th European Conference on Pattern Languages of Programs (EuroPLoP). pp. 1–35 (2021)
10. Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., Pautasso, C.: Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Addison-Wesley Signature Series (Vernon), Pearson Education (2023)