Swagger & OpenAPI 2.0 Quick Guide

Matthias Biehl

First edition: May 2018

Biehl, Matthias
API-University Press
Volume 9 of the API-University Series.
Includes illustrations, bibliographical references and index.
ISBN-13: 978-1719325202
ISBN-10: 1719325200

API-University Press
https://www.api-university.com
info@api-university.com

Contents

1 API Design **9**

1.1 API Design Decisions 9
1.2 API Frontend Design Decisions 10
1.3 Resources 11
1.4 URI Design 12
1.5 Representations 13
1.5.1 Content-Type of Representations 13
1.5.2 JSON 14
1.5.3 JSON Schema 14
1.6 Parameters 15
1.6.1 Use of Parameter Types 15
1.6.2 Parameter Types 16
1.6.2.1 Path Parameters 16
1.6.2.2 Query Parameters 16
1.6.2.3 Form Parameters 17
1.6.2.4 Header Parameters 18
1.7 Methods 20
1.7.1 GET 21
1.7.2 POST 21
1.7.3 PUT 22
1.7.4 DELETE 22
1.7.5 PATCH 23
1.7.6 HEAD 24
1.7.7 OPTIONS 24
1.7.8 TRACE 25
1.7.9 Non-Standard HTTP Methods 25

1.8 Status Codes 25
1.8.1 Overview of HTTP Status Codes 26
1.8.2 Redirection 27
1.8.3 Error Handling 28
1.8.3.1 Client Errors 28
1.8.3.2 Server Errors 30
1.8.3.3 Error Message 31
1.9 Input and Output Validation 32
1.9.1 Input Validation 32
1.9.2 Output Validation 33

2 API Description Languages 35
2.1 What are API Description Languages? 35
2.2 Usage 36
2.2.1 Communication and Documentation . . . 36
2.2.2 Design Repository 39
2.2.3 Contract Negotiation 39
2.2.4 API Implementation 40
2.2.5 Client Implementation 41
2.2.6 Discovery 41
2.2.7 Simulation 42
2.3 Language Features 42
2.4 Limitations 44

3 Swagger and OpenAPI 2.0 45
3.1 Overview 45
3.2 Root Element 47
3.3 Resources 49
3.4 Schema 51
3.5 Parameters 52
3.6 Reusable Elements 54
3.7 Security 55
3.7.1 Security Definition 55
3.7.2 Security Binding 56

Bibliography **61**

Abstract

Building RESTful APIs? Great - API consumers love them if the right design decisions were made. And we need to make a lot of design decisions when building a new API. How do we capture these decisions, gain an overview, communicate the decisions and iterate on the design?

To describe APIs in a precise manner, API designers typically use API description languages. They offer specialized language concepts to capture design decisions. Thus, API description languages can be called the power tools of the API designer.

One of the most popular API description languages is Swagger and OpenAPI. In this quick guide, we will learn how Swagger/OpenAPI can be used to capture your RESTful API design decisions.

1 API Design

Any type of design requires taking well-informed decisions. The decisions are intended to make the product better in some way, e.g. provide more functionality, provide better quality or a better user experience. Better design decisions typically lead to better products. This is no different when designing APIs. Which design decisions are there for APIs?

1.1 API Design Decisions

We see four groups of design decisions.

- **Architectural Design Decisions:** When designing an API, decisions have to be made regarding architectural issues, such as the patterns and the styles to be used. Should the API follow the REST or SOAP architectural style? These design decisions are foundational and have an impact on all following decisions.
- **API Frontend Design Decisions:** Since the frontend of the API is visible to the API consumers (= customers of the API), frontend design decisions are quite critical for the success of an API. Frontend design for APIs is typically RESTful design. For RESTful frontend design we need to answer questions such as: How does the URI of the API look like? Are the parameters passed in the form of query parameters or path parameters? Which headers and status codes should be used? We will address frontend design decisions for APIs in section 1.2.

- **API Backend Design Decisions:** The functionality of the API depends on leveraging data and services of backend systems. Backend design decisions address the connection between API and backend. Design decisions regarding the integration, transformation, aggregation, security and error handling of the backend have an impact on the functionality of the API.

- **Non-functional Design Decisions:** The architectural, frontend and backend design decisions are primarily taken to craft the functionality of the API. However, these decisions also have an impact on the non-functional properties of the API, such as security, performance, availability, and evolvability. Non-functional properties of the API should not be an afterthought. The API needs to be designed right from the start to fulfill non-functional requirements.

In this book we address API frontend design decisions.

1.2 API Frontend Design Decisions

The first thing we see about a new web application, is typically its user interface. This is why user-interface design is so important for web applications. The frontend of the API is what the consumer sees first when using the API. Now, user interface design is for web applications, what API frontend design is for APIs. The results of the frontend design are directly visible to the consumer of the API, who develops apps using the APIs. This is why the frontend design is said to have an effect on the developer experience (DX), in analogy to the user experience (UX) of user interfaces. In the eyes of developers, great API frontend design can turn good APIs into great APIs.

API frontend design depends on the architectural design decisions made earlier: Different design decisions have to made in

the API frontend whether the API follows the RPC, SOAP or REST architectural style. In this and the following chapters, we assume that the REST architectural style is chosen, since it is the most common and most popular architectural style for APIs. With this assumption we can focus exclusively on the design of RESTful APIs.

In this chapter we explain in detail how the architectural constraints of REST can be best used and applied for designing the frontend of APIs. We thus need to design the REST concepts, such as resources (see section 1.3), URIs (see section 1.4), representations (see section 1.5), parameters (see section 1.6), methods (see section 1.7) and status codes (see section 1.8). In addition we perform input and output validation (see section 1.9).

API frontend design lends itself to contract-first design. What does the contract specify? The API frontend design contract contains a specification of the endpoints, URLs, parameters, methods, the data models, data formats etc. Such a contract can be captured using API description languages (see chapter 2) such as Swagger (see chapter 3).

1.3 Resources

A resource model describes the structure of the data being served or being accepted by the API. Each API proxy has such a resource model – whether it is explicitly specified or not. Since it is hard to change the resource model later on, it is important to get it right from the beginning and invest some time in the design of the resource model.

Designing a resource model is similar to designing classes in object oriented design, with the difference that the methods of the resources are already predefined by the HTTP methods.

When designing a resource model, lots of questions need to