OpenAPI Ktor Generation Tool - OKGenTool

Hélio Fitas, n.º 39622, e-mail: A39622@alunos.isel.pt, heliofitas@gmail.com
Dinis Laranjeira, n.º 46081, e-mail: A46081@alunos.isel.pt, dinis.laranjeira14874@gmail.com

Supervisor: Pedro Félix, e-mail: pedro.felix@isel.pt

March 2024

1 Introduction

In this project proposal there are some key contextual topics that are important to clarify. The following paragraphs will explain the main context in detail.

1.1 HTTP APIs

Application Programming Interface (API) is a way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a interface is called an **API description** [1].

HTTP APIs are a type of interface that use the Hypertext Transfer Protocol (HTTP) to manage communications between clients and servers. They typically provide a means for clients to request information from servers, as well as a way for servers to send data back to clients.

1.2 OpenAPI

1.2.1 OpenAPI Specification

Accordingly to the OpenAPI Specification webpage [2]:

> "The **OpenAPI Specification** (OAS) defines a standard, programming language-agnostic interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network

traffic. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, the OpenAPI Specification removes guesswork in calling a service" .

1.2.2 OpenAPI Description

From the OpenAPI learning site [3]:

> "An **OpenAPI Description** (OAD) describes an HTTP-like API in one or more machine-readable documents (files or network resources). (...) OpenAPI Descriptions are written as one or more text documents. Each document represents a `JSON` object, in either `JSON` or `YAML` format. References are used to link parts of the `JSON` object(s) to each other, and this linked structure is the complete OpenAPI Description. Parsing begins with an OpenAPI Object, and the document containing that object is known as the entry document, commonly called `openapi.json` or `openapi.yaml`. (...) The root object in any OpenAPI Description is the OpenAPI Object, and only two of its fields are mandatory: `openapi` and `info`. Additionally, at least one of `paths`, `components` and `webhooks` is required.".

1.2.3 A Design-First Approach vs Code-First Approach

OpenAPI defines two different approaches for creating OADs. A Design-First approach and Code-First approach [4]:

- "In the Code-first approach, the API is first implemented in code, and then its description is created from it, using code comments, code annotations or simply written from scratch. This approach does not require developers to learn another language so it is usually regarded as the easiest one.
- Conversely, in Design-first, the API description is written first and then the code follows. The first obvious advantages are that the code already has a skeleton upon which to build, and that some tools can provide boilerplate code automatically."

OpenAPI Initiative (OAI) strongly suggests the use of **Design-first** approach. The reason is simple: The number of APIs that can be created in code is far superior to what can be described in OpenAPI. To emphasize: OpenAPI is not capable of describing every possible HTTP API, it has limitations.

1.3 Ktor

Ktor [5] is a framework for building asynchronous server-side and client-side applications, using Kotlin language [6]. Both Ktor and Kotlin are developed by JetBrains, creators of IntelliJ IDEA.

One of the benefits of Ktor and Kotlin is the support for multiplatform programming. It reduces time spent writing and maintaining the same code for different platforms while retaining the flexibility and benefits of native programming.

2 The Problem

In this context, tools can be created to generate the code needed to implement a defined API, ensuring that all transactions meets the specified requirements.

During our research, we encountered the tool *OpenAPI Generator* [7] that can generate code for both client and server side. However, when we test it we found a number of points that, could be improved. We found that the tool didn´t generated code for ktor multiplataform. Additionally the generated code wasn´t readonly, which limits the extend of the solution. Furthermore, this tool has a high complexity level making it harder to use.

Our tool will put in place this improvements adding value and differentiating it from the existing one.

3 Deliverables

At the end of this project we expect to deliver a user friendly command line tool that runs in JVM, generating code to implement an OAD for both servers and clients. The generated code acts as a library that programmers can call and use within their own code. The library generated by the command line tool should be interpreted by programmers as read-only code. Programmers should not edit the generated code directly.

This library will implement default behavior for each API endpoint, if the OAD provides one, allowing the programmer to implement only those endpoints for which they desire a specific implementation. If the OAD does not provides a default behavior, the programmer can call a method from the library to specify a default behavior for all unimplemented endpoints. If neither the OAD nor the programmer defines a default behavior, the library assumes that all unimplemented endpoints will return a HTTP response with a status code **501 Not implemented**.

The command-line tool can be run at any time it is needed without interfering with the programmer's code, for example, to automatically update the library when the description in the OAD changes. With this in mind, another output will be a Gradle plugin that allows the the programmer to regenerate the code every time they build the project, ensuring that the library is always updated with the last description version.

4 Risks and Challenges

Regarding the risks and challenges involved in this project, we have identified a few. The main risk is the complexity of the project taking into account it´s deadline. Another risk is our unfamiliarity with the Ktor framework that add a new layer of difficulty.

Lastly the complexity of the OpenAPI Specification makes it a risk to do all the features in such a small deadline, this will lead us to implement only the most common functionalities. This risks may lead to the need for extensive revisions of the initial design resulting in increased workload at each stage of the process.

In addition, we face a challenge in finding suitable libraries that can assist in the software development, as is necessary to find libraries compatible with ktor multiplatform and that offer the necessary functionalities.

5 Plan

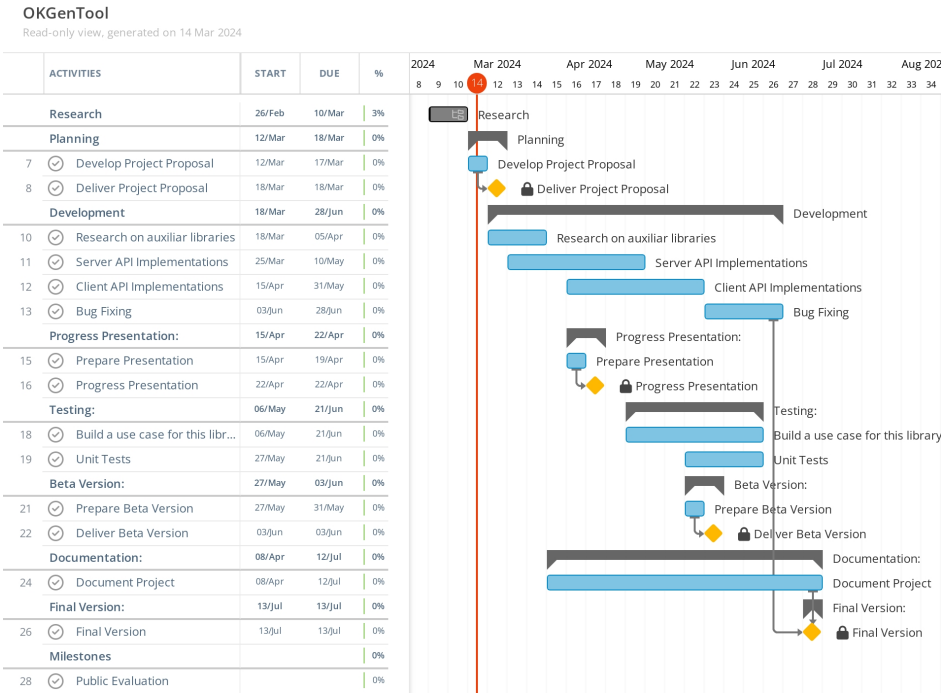

Figure 1: Project Plan

References

[1] Wikipedia contributors. API — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/API`, 2024. [Online; accessed 13-March-2024].

[2] OpenAPI Initiative. OpenAPI Specification v3.1.0. `https://spec.openapis.org/oas/latest.html`, February 2021. [Online; accessed 13-March-2024].

[3] OpenAPI Initiative. Structure of an OpenAPI Description. `https://learn.openapis.org/specification/structure`, 2023. [Online; accessed 13-March-2024].

[4] OpenAPI Initiative. Best Practices — Use a Design-First Approach. `https://learn.openapis.org/best-practices.html#use-a-design-first-approach`, 2023. [Online; accessed 13-March-2024].

[5] JetBrains. Ktor. `https://ktor.io/`, 2024. [Online; accessed 13-March-2024].

[6] JetBrains. Kotlin. `https://kotlinlang.org/`, 2024. [Online; accessed 13-March-2024].

[7] OpenAPI Generator. Source code. `https://github.com/OpenAPITools/openapi-generator`. [Online; accessed 18-March-2024].