REMOTE API WEB REFERENCE FOR JAVA ENTERPRISE APPLICATIONS

NÁZEV PRÁCE

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **Bc. ONDŘEJ KRPEC**
AUTOR PRÁCE

SUPERVISOR **Ing. RADEK KOČÍ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2017

Abstract

This thesis describes a draft of an application that is build on JRAPIDoc tool, which allows automatically generate an API documentation from an existing source code, but currently lacks its good and simple presentation. The theoretical part of the thesis explains the principles of web services, remote interfaces and examines the existing tools for automatic API generation and presentation. Subsequently, the design drafts of the application are presented in detail. The results of this thesis allows the development of the application, that will provide not only presentation of a generated API, but it will allow its testing through calling web services that are provided by the API. In conclusion this will lead to decrease need of funds for the development of the documentation and for the realization of the system integration.

Abstrakt

Tato práce popisuje návrh aplikace, která je postavena na základě nástroje JRAPIDoc, který umožňuje automatické generování API dokumentace ze zdrojového kódu, avšak postrádá její jednoduchou a srozumitelnou prezentaci. Teoretická část práce vysvětluje principy webových služeb, vzdálených rozhraní a zkoumá již existující nástroje pro automatické generování a zobrazení API. V práci jsou dále představeny designové návrhy aplikace a technologie, které budou použité při jejím vývoji. Výsledky této práce umožní vývoj aplikace, která poskytne automatické generování, zobrazení a testování API dokumentace na základě zdrojového kódu, což povede ke snížení nákladů potřebných pro tvorbu dokumentace a tím i realizaci systémové integrace.

Keywords

Remote API, Web Services, API documentation tool, System integration, JRAPIDoc, REST, Angular, PatternFly, Java Enterprise Applications, Red Hat

Klíčová slova

Vzdálená API, Webové služby, Nástroj na dokumentaci API, Systémová integrace, REST, JRAPIDoc, Angular, PatternFly, Red Hat

Reference

KRPEC, Ondřej. *Remote API Web Reference for Java Enterprise Applications.* Brno, 2017. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Kočí, Ph.D.

Remote API Web Reference for Java Enterprise Applications

Declaration

Hereby I declare that this term project was prepared as an original author's work under the supervision of Mr. Ing. Radek Kočí, Ph.D. The supplementary information was provided by Mr. Mrg. Ivo Bek. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Ondřej Krpec
December 4, 2017

Acknowledgements

I would like to thank Mr. Mgr. Ivo Bek for his technical leading of this Master Thesis. At the same time, I would like to thank Mr. Ing. Radek Kočí, Ph.D., for his pedagogical leadership.

Contents

Chapter 1

Introduction

In the software industry, the great and accessible code is crucial for modern business, and the best way for us to connect and share it is through APIs[1]. They are supposed to connect engineers, allow for the sharing of developments, let companies add value to the products and create an ecosystem of shared knowledge. To fulfill these tasks they have to be clear, accessible and most importantly human and machine readable. Despite their importance there hasn't been an industry standard for designing nor documentating them.

This thesis aim to solve the problem of documentating and presenting the APIs. The goal is to design an application having an innovative user interface with regard to clarity and simple use, aimed to developers, even in case of large interfaces. The application will be build upon existing JRAPIDoc [1] tool that provides a way to automate API generation, but currently lacks its good representation. Ultimately, the application will provide not only API reference, but also the functionality to test and call the listed web services.

The thesis is organized as follows. Chapter 2 gives definitions needed to follow the thesis and provides an overview of an existing API reference generators. Chapter 3 focuses on technologies that are going to be used for the implementation of the improvements. Chapter 4 describes the iterative application design that is based on live mockups and later on PatternFly Framework. The last Chapter contains an overall summary of the designed solution and final thoughts on the work done within the thesis.

[1]API is an abbreviation for an application programming interface which is a set of protocols and tools for building application software.

Chapter 2

Preliminaries And Definitions

This chapter will gradually introduce terms necessary to follow the thesis. In the first sections, we will provide explanation why is API documentation important and why it is important to automate API code generation. In the next section, we will go over the basic terminology and establish notion of remote interfaces. The last section covers the existing solutions for API code generation, their advantages and disadvantages.

2.1 What is API Documentation and Why It Matters?

In this section we explain what APIs are, we introduce the basic terms that relate to them and we focuses on why it is important to document them.

2.1.1 What is Application Programming Interface?

In computer programming an application programming interface is the defined interface through which interactions happen between an enterprise and users of its assets. It can become the primary entry point for enterprise service, for its own website and applications, as well as for a partner and customer integrations. It is defined through a contract so that any application can use it with relative ease.

The API approach creates a loosely coupled architecture that allows a component service to have a wide range of future uses, and is technology agnostic. The architecture resolves around providing programmable interfaces to a set of services to different applications serving different kinds of customers. It assumes that these user groups might change or evolve over time in the way they utilize the provided services. The strategy of providing APIs leads to the following benefits:

- **Automation**. With APIs, computers rather than people can manage the work. Through them, companies can update work flows to make them quicker and more productive.
- **Integration**. They allow content to be embedded from any site or application more easily. This guarantees more fluid information delivery and an integrated user experience.
- **Personalization**. Any user or company can customize the content and services that they use the most.

- **Reduction of costs**. They represent a cheaper way of building applications by increasing the reuse of services. Providing a usage or "analytics-based" evolutionary development platform decreases cost of development and change to services.
- **Increasing customer loyalty**. The company that releases the API allows its customers to access their conferencing services in new, more efficient ways, increasing brand recognition and customer loyalty.

Overall, they allow users to enhance and add services over existing products. This introduces the product to a new type of user: "the third-party" developer. However catering to the developers is tricky. They want to understand how to use the API quickly and effectively, which is why API documentation comes into the picture.

2.1.2 What is API Documentation and Why We Need One?

API documetation is a concise reference manual that contains all the information required to work with the API, aiming to cover everything a "third-party" developer would need to know to use it. To facilitate understanding, the documentation can include details about the functions, classes, return types and more, supported by tutorials and examples. However, implementation details of the provided services are usually omitted. Restrictions and limitations on how the API can be used can also be covered by the documentation, for example the documentation of a function could note that its parameters cannot be null, or that the function itself is not thread safe.

When using the API the "third-party" developer, who is API's main consumer, want to integrate as quickly as possible to move forward in his development, meaning he should immediately understand the value and usage of the API. The aggregate experience of the developer when discovering, learning to use, and finally integrating with the API is termed as Developer Experience [2]. Generally speaking, it is essentially the sum of the experiences using the API. If many of those experiences were difficult or just annoying, the developers are going to have a bad experience and they won't want to use the services that the API provides. This implies that good documentation is the key to a great DX, because even the best, functional service, will be unused if the developers don't know how to use it. On the contrary as more users adopt the provided services and reach critical mass, there will be a probable increase in publicity, leading to a network effect[1].

In addition to driving increased awareness and adoption of the provided services, good documentation also leads to easier product maintenance. It helps internal teams know the details of resources, methods, and their associated requests and responses, making maintenance and updates quicker and cheaper.

2.1.3 Why Automate API Code Generation?

Among all the phases in the API lifecycle, documentation is probably the area showing the most growth. However the developers pay little to no attention to it when launching code. For a lot of developers it is because writing documentation is still tedious and time consuming task. It is because the documentation is so comprehensive, it can be difficult for them to keep it updated. This is especially true for teams that rely on static documentation that is manually updated with every release of new version of the API. Therefore

[1]A network effect is the positive effect described in economics and business that an additional user of a good or service has on the value of that product to others.

the more we can automate around the documentation, the less work it is for the developers to iterate and make improvements. That is why there is a great need of a framework that allows "auto-generate" the documentation that developers and team can customize, and build on to create a more comprehensive user manual for working with the API.

2.2 Understanding Web Services

A web service is a software system designed to support interoperable machine to machine interaction over a network. It is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to interprocess communication on a single computer. This interoperability is due to the use of open standards.

// TODO link na soap In the past, web services used mostly SOAP over HTTP protocol, allowing less costly interactions over the internet than via proprietary solutions like EDI/B2B. In a 2004, the W3C extended the definition of web services about REST-complient web services, in which the primary purpose of the web service is to manipulate XML representations of web resources using a uniform set of stateless operations.

2.2.1 "SOAP-based" Web Services

SOAP is an acronym for Simple Object Access Protocol. It's a "XML-based" messaging protocol for exchanging information among computers that uses WSDL TODO poznamka pod carou wsdl is an XML-based protocol that describes how to access a web service and what operation it will perform**??** for communication between consumer and provider of the web service. Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.

A SOAP message is an ordinary XML document containing the following elements:

- **Envelope**. Defines the start and the end of the message. Envelope is a mandatory element.
- **Header**. Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate "end-point". Header is an optional element.
- **Body**. Contains the XML data comprising the message being sent. Body is a mandatory element.
- **Fault**. An optional Fault element that provides information about errors that occur while processing the message.

As shown in listing 2.1, a SOAP message consists of an Envelope element, inside which a Header and a Body element can be nested. Inside the Body element a Fault element can be nested, if an error occurs in the web service. This SOAP message format can be used both to send requests from the client to the web service, and to send responses back to the client from the web service. Thus the SOAP request and response message format is the same unlike the HTTP protocol where the request and response formats are different.

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    <!-- Fault element is optional,
       used only if a fault occurs in a web service -->
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Listing 2.1: Example of SOAP Message Structure.

When a client and a web service communicate via SOAP, they exchange messages. In the SOAP specification **??** are described two basic message exchange patterns.

- Request - Response
- Response

Request - Response pattern

In a request - response message exchange the SOAP client sends a SOAP request message to the web service. The service responds with a SOAP response message.

A request - response message exchange pattern is necessary when the SOAP client needs to send data to the SOAP service. For instance, if the client request that data be stored by the service, the client needs to send the data to be stored to the service.

// TODO obrazek example

Response pattern

In a response message exchange the SOAP client connects to the service, but does not send a SOAP message through. It just sends a simple HTTP request for instance. The service responds with a SOAP message. This pattern is similar to how browser communicates with a web server. The browser sends an HTTP request telling what page it wants to access, but the HTTP request does not carry any additional data. The web server responds with an HTTP response carrying the requested page.

// TODO obrazek example

2.2.2 RESTful Web Services

2.3 Existing solutions for API code generation

2.3.1 JRAPIDoc

2.3.2 Swagger

2.3.3 Apiary

2.3.4 Spring REST Docs

2.3.5 Postman

https://nordicapis.com/ultimate-guide-to-30-api-documentation-solutions/

Chapter 3

Technologic overview

3.1 Angular

3.2 TypeScript

3.3 PatternFly

Chapter 4

Application Design

4.1 The initial state

4.2 Building live mockups

4.3 Improving the mockups with PatternFly

Chapter 5

Conclusion

Bibliography

[1] Jiříček, T.: Nástroj pro generování dokumentace vzdálených rozhraní (API) webových služeb aplikací Java Enterprise Edition [diplomová práce]. FEL ČVUT v Praze. 2015.

[2] Rangel, F.: API for Humans: The Rise of Developer Experience (DX). September 2015. [Online; visited 17.11.2017].
Retrieved from:
https://www.hellosign.com/blog/the-rise-of-developer-experience