



# Pro RESTful APIs with Micronaut

Build Java-Based Microservices with  
REST, JSON, and XML

—  
*Third Edition*

—  
Sanjay Patni

Apress®

# **Pro RESTful APIs with Micronaut**

**Build Java-Based Microservices  
with REST, JSON, and XML**

**Third Edition**

**Sanjay Patni**

**Apress®**

# ***Pro RESTful APIs with Micronaut: Build Java-Based Microservices with REST, JSON, and XML, Third Edition***

Sanjay Patni  
Milpitas, CA, USA

ISBN-13 (pbk): 979-8-8688-1242-2  
<https://doi.org/10.1007/979-8-8688-1243-9>

ISBN-13 (electronic): 979-8-8688-1243-9

Copyright © 2025 by Sanjay Patni

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Melissa Duffy  
Development Editor: Laura Berendson  
Editorial Assistant: Gryffin Winkler

Cover designed by eStudioCalamar

Cover image designed by FlyD on Unsplash

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit <https://www.apress.com/gp/services/source-code>.

If disposing of this product, please recycle the paper

*I would like to thank everyone at Apress who I have worked closely with. Thanks to the reviewers, their in-depth reviews helped the quality of the book. A heartfelt thanks goes to my wife, Veena, for her tireless and unconditional support that helped me work on this book. A huge thanks goes to my father, Ajit Kumar Patni, and my mother, late Basantidevi, for their selfless support that helped me reach where I am today.*

# Table of Contents

- About the Author .....xiii**
- About the Technical Reviewer .....xv**
- Introduction .....xvii**
  
- Chapter 1: Fundamentals of RESTful APIs ..... 1**
  - SOAP vs. REST ..... 4
  - Web Architectural Style..... 6
    - Client-Server ..... 7
    - Uniform Resource Interface..... 7
    - Layered System..... 7
    - Caching..... 8
    - Stateless..... 8
    - Code-on-Demand ..... 8
    - HATEOAS..... 9
  - What Is REST?..... 10
    - REST Basics..... 11
    - REST Fundamentals ..... 12
  - Summary..... 14
  
- Chapter 2: Micronaut..... 15**
  - Comparison of Micronaut with Spring Boot..... 16
    - Ease of Installation ..... 16
    - Natively Cloud Enabled..... 17

## TABLE OF CONTENTS

Serverless Functions .....	17
Application Configuration .....	18
Messaging System Support.....	19
Security .....	20
Caching.....	21
Management and Monitoring .....	21
API Portfolio .....	22
Online Flight .....	22
Message .....	23
Software.....	24
Micronaut .....	24
JDK 21 .....	25
POSTMAN .....	25
CURL.....	25
IDE .....	25
Maven.....	26
Summary.....	27
<b>Chapter 3: Introduction: XML and JSON .....</b>	<b>29</b>
What Is XML? .....	29
XML Comments .....	31
Why Is XML Important?.....	32
How Can You Use XML?.....	33
Pros and Cons of XML.....	33
What Is JSON? .....	34
JSON Syntax.....	35
Why Is JSON Important? .....	37
How Can You Use JSON? .....	38

Pros and Cons of JSON..... 38

XML and JSON Comparison..... 39

Implementing APIs to Return XML and JSON Messages..... 41

Summary..... 45

**Chapter 4: API Design and Modeling ..... 47**

API Design Strategies..... 47

API Creation Process and Methodology ..... 49

    Process..... 50

    API Methodology..... 50

    Domain Analysis or API Description..... 51

    Architecture Design ..... 52

    Prototyping ..... 53

    Implementation ..... 53

    Publish..... 54

    API Modeling..... 54

    Comparison of API Modeling..... 56

    In Summary ..... 57

Best Practices ..... 58

    Keep Your Base URL Simple and Intuitive..... 58

    In Summary ..... 59

    Error Handling ..... 60

    Error Code..... 61

    Versioning..... 62

    Partial Response..... 63

    Pagination..... 64

    Multiple Formats ..... 64

    API Façade..... 65

## TABLE OF CONTENTS

API Solution Architecture .....	65
Mobile Solutions.....	66
Cloud Solutions.....	67
Web Solutions.....	67
Integration Solutions .....	67
Multichannel Solutions .....	67
Smart TV Solutions .....	68
Internet of Things .....	68
Stakeholders in API Solutions .....	68
API Providers .....	68
API Consumers .....	68
End Users .....	69
API Modeling .....	69
OpenAPI (Swagger).....	69
Summary.....	76
<b>Chapter 5: Introduction to JAX-RS .....</b>	<b>77</b>
JAX-RS Introduction.....	77
Input and Output Content Type.....	79
JAX-RS Injection .....	79
Path Parameter .....	81
Query Parameter .....	81
Cookie Parameter .....	82
Header Parameter .....	82
Form Parameter .....	82
Matrix Parameter .....	82
Micronaut Implementation of JAX-RS.....	83
Supported Annotations.....	84



Injectable Parameter Types.....	85
SecurityContext and Micronaut Security.....	86
Summary.....	86
<b>Chapter 6: API Portfolio and Framework .....</b>	<b>87</b>
API Portfolio Architecture .....	87
Requirements .....	87
Consistency .....	88
Reuse.....	88
Customization .....	88
Discoverability .....	89
Longevity .....	89
How Do We Enforce These Requirements—Governance? .....	89
Consistency .....	89
Reuse.....	90
Customization .....	90
Discoverability .....	90
Change Management .....	90
API Framework.....	91
Process APIs: Services Layer.....	93
System APIs: Data Access Object .....	93
Experience APIs: API Façade.....	93
Services Layer Implementation .....	94
Summary.....	101
<b>Chapter 7: API Platform and Data Handler .....</b>	<b>103</b>
API Platform Architecture.....	103
Why Do We Need an API Platform? .....	104
So What Is an API Platform?.....	104

TABLE OF CONTENTS

So Which Capabilities Does the API Platform Have? ..... 105

API Development Platform ..... 105

API Runtime Platform ..... 107

API Engagement Platform ..... 107

How Is an API Platform Organized? What Is the Architecture of the  
API Platform? ..... 108

How Does the API Architecture Fit in the Surrounding Technical  
Architecture of an Enterprise? ..... 110

Data Handler ..... 112

Data Access Object ..... 112

Command Query Responsibility Segregation (CQRS)..... 113

SQL Development Process ..... 113

NoSQL Process..... 114

Do I Have to Choose Between SQL and NoSQL? ..... 114

Why a Single REST API? ..... 115

Summary..... 133

**Chapter 8: API Management and CORS ..... 135**

    Façade ..... 135

    Façade Pattern..... 135

    API Façade ..... 136

    API Management..... 139

    API Life Cycle ..... 140

    API Retirement ..... 141

    API Monetization ..... 142

    Cross-Origin Resource Sharing (CORS)..... 143

    Summary..... 143

<b>Chapter 9: API Security .....</b>	<b>145</b>
API Security—OAuth 2 .....	145
Roles .....	146
Tokens.....	146
Register as a Client.....	148
Client Registration.....	148
Authorization Server Response.....	149
Authorization Grant Types .....	149
Authorization Code Grant .....	149
When Should It Be Used?.....	149
Implicit Grant Flow .....	151
When Should It Be Used?.....	151
Resource Owner Password Credentials Grant .....	153
When Should It Be Used?.....	154
Client Credentials Grant .....	155
API Security—JSON Web Token.....	157
Summary.....	162
<b>Index.....</b>	<b>163</b>

# About the Author



**Sanjay Patni** is a results-focused technologist with extensive experience in aligning innovative technology solutions with business needs to optimize manual steps in the business processes and improving operational efficiency.

At Oracle, he has worked with the Fusion Apps Product development team, where he has identified opportunities for automation of programs related to Fusion Apps codeline management. This involved delivery of GA releases for patching, as well as codelines for ongoing demo, development, and testing. He conceptualized and developed self-service UX for codeline requests and auditing, reducing manual steps by 80%. He also rolled out 12 sprints of codeline creation, automating about 100+ manual steps involving integration with other subsystems using technologies like automation workflow and RESTful APIs.

Prior to joining Oracle, he spent 15+ years in the software industry, defining and delivering key initiatives across different industry sectors. His responsibilities included innovation, requirement, analysis, technical architecture, design, and agile software development of web-based enterprise products and solutions. He pioneered innovative usage of Java in building business applications and received an award from Sun Microsystems. This helped improve feedback for Java APIs for Enterprise in building business application software using Java. He has diverse experience in application architecture including UX, distributed systems, and cloud.

## ABOUT THE AUTHOR

He has worked as a visiting technical instructor or mentor and conducted classes or training on RESTful API design and integration.

He has a strong educational background in computer science with a master's from IIT, Roorkee, India and bachelor's in Electronics from SGSITS, Indore, India.

# About the Technical Reviewer



**Massimo Nardone** has more than 26 years of experience in security, web/mobile development, and cloud and IT architecture. His true IT passions are security and Android. He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 25 years. He holds a Master of Science degree in Computing Science from the University of Salerno, Italy. He has worked as a chief information security officer (CISO), software engineer, chief security architect, security executive, and OT/IoT/IIoT security leader and architect for many years.

# Introduction

Databases, websites, and business applications need to exchange data. This is accomplished by defining standard data formats such as Extensible Markup Language (XML) or JavaScript Object Notation (JSON), as well as transfer protocols or web services such as the Simple Object Access Protocol (SOAP) or the more popular Representational State Transfer (REST). Developers often have to design their own Application Programming Interfaces (APIs) to make applications work while integrating specific business logic around operating systems or servers. This book introduces these concepts with a focus on RESTful APIs.

This book introduces the data exchange mechanism and common data formats. For web exchange, you will learn the HTTP protocol, including how to use XML. This book compares SOAP and REST and then covers the concepts of stateless transfer. It introduces software API design and best design practices. The second half of the book focuses on RESTful API design and implementations that follow the Micronaut and Java API for RESTful web services. You will learn how to build and consume Micronaut services using JSON and XML and integrate RESTful APIs with different data sources like relational databases and NoSQL databases through hands-on exercises. You will apply these best practices to complete a design review of publicly available APIs with a small-scale software system in order to design and implement RESTful APIs.

This book is intended for software developers who use data in projects. It is also useful for data professionals who need to understand the methods of data exchange and how to interact with business applications. Java programming experience is required for the exercises.

## INTRODUCTION

Topics covered in this book include

Data exchange and web services

SOAP vs. REST, state vs. stateless

XML vs. JSON

Introduction to API design: REST and Micronaut

API design practices

Designing RESTful APIs

Building RESTful APIs

Interacting with RDBMS (MySQL)

Consuming RESTful APIs (i.e., JSON and XML)



## CHAPTER 1

# Fundamentals of RESTful APIs

APIs are not new. They've served as interfaces that enable applications to communicate with each other for decades. But the role of APIs has changed dramatically in the last few years. Innovative companies have discovered that APIs can be used as an interface to the business, allowing them to monetize digital assets, extend their value proposition with partner-delivered capabilities, and connect to customers across channels and devices. When you create an API, you are allowing others within or outside of your organization to make use of your service or product to create new applications, attract customers, or expand their business. Internal APIs enhance the productivity of development teams by maximizing reusability and enforcing consistency in new applications. Public APIs can add value to your business by allowing third-party developers to enhance your services or bring their customers to you. As developers find new applications for your services and data, a network effect occurs, delivering significant bottom-line business impact. For example, Expedia opened up their travel booking services to partners through an API to launch the Expedia Affiliate Network, building a new revenue stream that now contributes \$2B in annual revenue. Salesforce released APIs to enable partners to extend the capabilities of their platform and now generates half of their annual revenue through those APIs, which could be SOAP-based (JAX-WS) and, more recently, RESTful (JAX-RS), Spring Boot, and now Micronaut.

SOAP web service depends upon a number of technologies (such as UDDI, WSDL, SOAP, and HTTP) and protocols to transport and transform data between a service provider and the consumer and can be created with JAX-WS.

Later, Roy Fielding (in the year 2000) presented his doctoral dissertation, “Architectural Styles and the Design of Network-based Software Architecture.” He coined the term “REST,” an architectural style for distributed hypermedia systems. Put simply, REST (short for Representational State Transfer) is an architectural style defined to help create and organize distributed systems. The key word from that definition should be “style,” because an important aspect of REST (and which is one of the main reasons books like this one exist) is that it is an architectural style—not a guideline, not a standard, or anything that would imply that there are a set of hard rules to follow in order to end up having a RESTful architecture.

In this chapter, I’ll be covering REST fundamentals, SOAP vs. REST, and web architectural style to provide a solid foundation and better prepare you for what you’ll see in later chapters.

The main idea behind REST is that a distributed system, organized RESTfully, will improve in the following areas:

- **Performance:** The communication style proposed by REST is meant to be efficient and simple, allowing a performance boost on systems that adopt it.
- **Scalability of component interaction:** Any distributed system should be able to handle this aspect well enough, and the simple interaction proposed by REST greatly allows for this.
- **Simplicity of interface:** A simple interface allows for simpler interactions between systems, which in turn can grant benefits like the ones previously mentioned.

- **Modifiability of components:** The distributed nature of the system, and the separation of concerns proposed by REST (more on this in a bit), allows for components to be modified independently of each other at a minimum cost and risk.
- **Portability:** REST is technology- and language-agnostic, meaning that it can be implemented and consumed by any type of technology (there are some constraints that I'll go over in a bit, but no specific technology is enforced).
- **Reliability:** The stateless constraint proposed by REST (more on this later) allows for the easier recovery of a system after failure.
- **Visibility:** Again, the stateless constraint proposed has the added full state of said request (this will become clear once I talk about the constraints in a bit). From this list, some direct benefits can be extrapolated. A component-centric design allows you to make systems that are very fault-tolerant. Having the failure of one component not affecting the entire stability of the system is a great benefit for any system. Interconnecting components is quite easy, minimizing the risks when adding new features or scaling up or down. A system designed with REST in mind will be accessible to a wider audience, thanks to its portability (as described earlier). With a generic interface, the system can be used by a wider range of developers. In order to achieve these properties and benefits, a set of constraints were added to REST to help define a uniform connector interface. REST is not suggested to be used when you need to enforce a strict contract between the client and server and when performing transactions that involve multiple calls.

## SOAP vs. REST

Table 1-1 provides a comparison between SOAP and REST with an example of use cases each can support.

**Table 1-1.** SOAP vs. REST comparison

Topic	SOAP	REST
Origin	SOAP (Simple Object Access Protocol) was created in 1998 by Dave Winer et al. in collaboration with Microsoft. Developed by a large software company, this protocol addresses the goal of addressing the needs of the enterprise market	REST (Representational State Transfer) was created in 2000 by Roy Fielding at UC, Irvine. Developed in an academic environment, this protocol embraces the philosophy of the Open Web
Basic concept	Makes data available as services (verb + noun), for example, “getUser” or “PayInvoice”	Makes data available as resources (nouns), for example, “user” or “invoice”
Pros	Follows a formal enterprise approach Works on top of any communication protocol, even asynchronously Information about objects is communicated to clients Security and authorization are part of the protocol Can be fully described using WSDL	Follows the philosophy of the Open Web Relatively easy to implement and maintain Clearly separates client and server implementations Communication isn’t controlled by a single entity Information can be stored by the client to prevent multiple calls Can return data in multiple formats (JSON, XML, etc.)

*(continued)*