Practical hapi

Build Your Own hapi Apps and Learn from Industry Case Studies

—

Kanika Sud

Apress®

Practical hapi

Build Your Own hapi Apps and Learn from Industry Case Studies

Kanika Sud

Apress®

Practical hapi

Kanika Sud
Kanika, Chandigarh, India

ISBN-13 (pbk): 978-1-4842-5804-0 ISBN-13 (electronic): 978-1-4842-5805-7
https://doi.org/10.1007/978-1-4842-5805-7

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit 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 rights@apress.com, or visit http://www.apress.com/rights-permissions.

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 via the book's product page, located at www.apress.com/9781484258040. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Dedicated to the Turbo C editor that introduced me to programming and to my parents, for always watching my back.

Table of Contents

About the Author

Kanika Sud has been working on the Web for over 10 years now. Her work spans enterprise CMSs in Java and backend technologies in LAMP stack and MEAN stack. She has also worked on open source ecommerce CMSs and UX strategy. Solution design remains her key favorite, while market research on mobile apps and plugins led her to experiment with a bootstrapped startup in technology, called Codnostic Solutions. Find her on LinkedIn: www.linkedin.com/in/kanikasud.

About the Technical Reviewer

Alexander Chinedu Nnakwue has a background in mechanical engineering from the University of Ibadan, Nigeria, and has been a front-end developer for over 3 years working on both web and mobile technologies. He also has experience as a technical author, writer, and reviewer. He enjoys programming for the Web, and occasionally, you can also find him playing soccer. He was born in Benin City and is currently based in Lagos, Nigeria.

Acknowledgments

I am indebted to my parents for the kind of support I've received all my life. It's a privilege to be supported by members of the family when you're busy working on your dreams.

This is in loving memory of my mother, who wanted me to live my dreams, all my life. I've also had the privilege to work with some of the best in the industry, whose collaboration and inspiration laid the road ahead for my learning. This book, and hopefully many more to come, is because of the best in business – Apress, and their wonderful team including, but not limited to, James, Nancy, Louise, and Alexander. Their patience, guidance, and step-by-step feedback for the narrative and technical assessment were invaluable.

Introduction

Research – that's what brought me to hapi. Flexibility, maintainability, and scalability are core areas when choosing a framework. I was new to Node.js, and I thought any framework I chose would be a huge learning curve. hapi was a pleasant surprise, because it gets you up and running in no time. If you're new to JavaScript, worry not – we've covered the ground well. This book is to make sure that you work from the ground up and have a solid foundation in hapi, and if you're familiar with JavaScript, you can always revisit the language and take a walk through advanced concepts like promises and others. We haven't covered any concepts like introducing SSL, node inspector, CORS, and so on. These are independent topics, and various informative articles about these are found on the Internet. What we've focused on is building a story out of a framework, getting all the major corners secured, so that the eager developer is confident enough to explore more and has his fundamentals right.

Before you step into code and make apps, services, and/or your next big project, be sure of "why" you wish to learn coding. That might not seem the right fit for an introduction to a technical book, but to all my learners, the joy of coding can really be assessed if you're sure of why you're learning, and before you get caught in an industry where timelines can get the better of you, it's only fair to yourself to keep the spark of research alive.

CHAPTER 1

Understanding REST APIs

This chapter throws light on building representational state transfer (REST)-based services. Since we'll be learning RESTful Web Services through our chosen language, Node.js, and using an awesome framework like hapi, it's first necessary to introduce ourselves to REST APIs and then understand how hapi makes the process easy. Along the ride, we'll learn about HTTP verbs, resource handling, stateless constraints, and tying it all into representational state transfer.

First Steps

If you really need to know why we're all talking about APIs in the early twenty-first century, you need to understand that while developed software is what the end user sees, at the back end, it's really data exchange. And who is exchanging that data? Two or multiple entities and layers of software. What is the format of such data? XML was a very popular data exchange format in the early stages of data talk – and while we look into the reasons of using JSON as a data exchange format in our capstone project later, we need to understand how typical data exchange looks and how it's different from the presentation layer.

Consider a scenario where we need information about how many students signed in to the chemistry lab on a particular date. Let's assume that the information is in a register with the chemistry lab manager. So what you're doing is asking for a data exchange between that register and yourself, via the lab manager. What does the lab manager do? He serves the information (that's a service) and provides information in the format you want (that's the data set returned) and the medium, in this case would be whatever mechanism he chooses to filter such information from the register. Fortunately for us, the world moved away from paper to digital long ago; and instead of registers and books, we came up with the programmable Web.

© Kanika Sud 2020
K. Sud, *Practical hapi*, https://doi.org/10.1007/978-1-4842-5805-7_1

API

The term API means an application programming interface, where an interface is a medium of communication. When we exchange data, our medium is a service that will use logic to talk to a database or another application layer for fetching data.

If you've spent a fair share of your time in web programming, you already know what the request and response cycle is about. For those who've migrated from other backgrounds, all you're expected to know is that all of the Web is ultimately a request and response cycle from and to a server. So typically, one caller – the "client" – calls out to the "server" and asks for data sets, and the latter serves the request and returns a response in an object over a protocol, a set of predefined rules of communication. After all, all we're doing is talking, aren't we?

REST API

REST APIs and RESTful Web Services made through REST APIs rely on an architectural style called representational state transfer. If you're a beginner, just remember that you are representing the state of an object or a resource and transferring it over the Web. It becomes more of a contract while we are dealing with resource exchange, with a set of rules.

The Web ultimately is a hub of resources. Remember the example of the lab manager? You're asking for resource information when you want data of all students who signed in on a particular date. Such resource information given by services on the Web needs to be scalable, easy to implement, maintainable, and extensible because it's not going to be a one-time request like our real-world example of the lab manager, hence the emphasis on design. A RESTful design performs extremely well where all you need is data to be consumed by a consumer – this consumer can be any client software, for instance, a mobile app. All your handheld apps request some resources, and those resources are served by hitting REST API URLs.

Now there are a few insights, two of which are pretty straightforward:

- The mobile app that is requesting the data should be able to understand the format in which the response is given.
- Breaking the representation tactfully into smaller resources is a good way of creating the response and utilizing it.