Extracted from:

Design and Build Great Web APIs

Robust, Reliable, and Resilient

This PDF file contains pages extracted from *Design and Build Great Web APIs*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2020 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

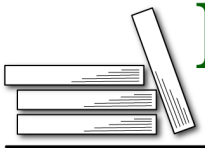
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

The Pragmatic Bookshelf

Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Copy Editor: Molly McBeath
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Copyright © 2020 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 978-1-68050-680-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—October 2020

Adopting the API-First Principle

The concept of API First has a long and varied history. Although the phrase has been in use for almost ten years (as of this writing), it's become particularly popular over the last several years. For example, the website ProgrammableWeb ran a series called "Understanding API First Design" in 2016,[1] and I still hear people mention this series when they talk about API First. A website created in 2013 was even dedicated to API First,[2] although it hasn't been maintained over the years. And a simple web search on the term "API First" brings up lots of related articles from all corners of the IT world targeted to a wide audience, from developers to CEOs.

I first heard about API First in 2009 in a blog post called "API First Design," by Kas Thomas.[3] He described the idea this way: "API-first design means identifying and/or defining key actors and personas, determining what those actors and personas expect to be able to do with APIs." I think this definition is important because it points out that creating APIs is about understanding who's using the API and what they want to *do* with it. Frankly, it's pretty hard to go wrong with this point of view in mind.

Another key element wrapped up in the API First meme is the notion of designing the API *before* you get excited about all the implementation details, such as programming languages, protocols, URLs, and message formats. Well-known API advocate Kin Lane puts it directly in his blog post "What Is an API First Strategy?"[4] when he states, "Before you build your website, web, mobile or single page application you develop an API first."

So, two key elements of the API-First principle are (1) focus on who's using the API and what they want to accomplish and (2) design the API *first*, before you start to think about technology details. A third important element in all this pertains to the business side of things. APIs exist to solve business problems. Let's explore these three elements a bit more to get a solid footing on the API-First principle, starting with how APIs are used to solve business problems.

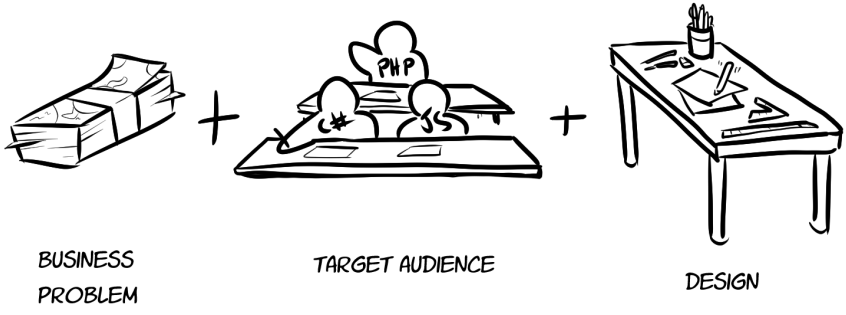
Using APIs to Solve Business Problems

In the real world, APIs exist to solve a business problem, like improving service quality, increasing sales, reducing costs, and so forth. When you're working

1. https://www.programmableweb.com/api-university/understanding-api-first-design
2. http://api-first.com
3. http://asserttrue.blogspot.com/2009/04/api-first-design.html
4. https://apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-to-this-new-question

• Click HERE to purchase this book now. discuss

on an API—whether you're tasked with designing it from scratch or simply working to implement a design someone else has provided—it helps to keep the business goals in mind.



Typically, APIs are needed to solve problems in three basic categories:

- Reducing the time/cost of getting something done
- Increasing the ease or likelihood of getting something done
- Solving a problem no one else has solved

To solve the first problem (how to reduce time/cost), APIs are often used behind the scenes—between teams in the same company, for example—in ways that are meant to automate some process or bridge a gap between one team (or software component) and another. In these cases, the design needs to be specific enough to solve the problem and flexible enough to be modified in the future without causing more pain for the teams involved. In other words, you're providing a short-term fix with long-term support in mind. That mindset changes the way you design and implement the solution and is the basis of the API we'll build in this book.

To solve the second problem (how to increase the ease), APIs are usually used to improve throughput or increase sales, user sign-ups, and so on. These APIs are aimed at making developers more productive as well as making it easier to solve interaction problems for humans. That often means APIs that support user experiences (UX) and are geared toward supporting mobile or web user interfaces (UIs). Supporting a live UI is a tough job. UI designers often have very specific demands on how the interfaces "look and feel" and need APIs that make that quick and easy. When you are supporting APIs for UX designers, the most important thing is getting them the data they need at the right moment. Often these APIs are quite "chatty" and have lots of variations and nuances. We won't spend too much time on UX-style APIs in this book, but I'll cover some considerations later in the book when we deploy and then modify an existing API.

• Click HERE to purchase this book now. discuss

Finally, you might be asked to build APIs that provide a solution to a problem no one has yet solved. These can be fun APIs to work on, but they have their own challenges. Most of the time, all the functional elements needed to solve the problem already exist within the company (for example, existing services, third-party products, remote web APIs, and so on). However, no one has figured out the best way to *mix* all these disparate components into a single solution. Depending on your company's IT ecosystem, this might be the most common work you need to do. And it might be the most frustrating. A lot of your time might be spent translating output from one service into something another service can understand. And if one of the existing services changes their output, your solution might fail. Designing and implementing this kind of *composed* API solution takes lots of attention to details and planning for possible failure. I'll spend some time in the second part of the book talking about how to deal with this kind of API challenge.

Designing APIs for People

As mentioned at the start of this chapter, a key element of API First is understanding the people using the API and the reason they need it. It's important to have a clear sense of the expected skills and interests of your API *consumer* (the person using the API) as well as the actual work they're trying to do (such as automate a process, support a UX, solve unique problems, and so on). Getting a good sense of these aspects of the API will help you spend time focusing on the right things when it comes to translating their needs into your design.

For example, are most of your API consumers working within your company? If they are, they likely already understand your company's business model, internal acronyms, and other "inside" information. Do most of them work in IT? If yes, you can assume they have some programming skills and understand things like HTTP, JavaScript, and PHP. Or maybe they work in marketing, human resources, or sales. They might not have extensive developer training or have much time to spend testing and debugging APIs. In this case, your API design needs to be straightforward and easy to use. You'll design and document the API differently depending on who you expect to use it.

If the target audience for your API is someone outside the company (such as a partner company or some third-party mobile developer), your approach will need to be different. You can't assume the developers understand (or even *care about*) your business model. In fact, you probably don't want to share too much "inside" information with them—they might be a competitor! In this case, you'll design and document an API that solves a vary narrow set of

• Click HERE to purchase this book now. discuss

problems and makes it hard for API consumers to mistakenly or maliciously use the API in a way that does harm to your company, data, or users.

Knowing the target audience of your API and the kinds of problems those people want to solve is critical when it comes to creating APIs that meet the needs of your audience without jeopardizing the safety and security of your own company.

Understanding APIs and Design

Finally, the way you design the API itself depends on several factors. The API-First principle tells us to design APIs without regard to the technology you might use to build it. That means we focus on the interactions in the API experience as well as the data behind it. In fact, in well-designed APIs, the interaction is paramount. It's important to first determine what tasks need to be done. Only then do you spend time figuring out just what data you need to pass back and forth for each interaction in order to accomplish the required tasks. I'll spend time talking about a design strategy in depth in Part II.

A really good process for supporting API-First design is to gather and then document the list of actions (for example, `FindCustomerRecord`, `UpdateCustomerHistory`, `RemoveCustomerHold`). Once you have a complete list, go back and start to list all the data properties (`customerId`, `givenName`, `familyName`, `email`, `shoeSize`, and so on) you need to share for each interaction. However, collecting and detailing the interactions is just the start. The real design work begins *after* you gather all this information.

In some cases, your API consumers will have all the data required ahead of time (`customerId`, `givenName`, `familyName`, `email`, and so on) and just need to plug values into your API and execute a single call. In other cases, only part of the required data is available (such as `email`). When your API consumers don't have all the data, they need helper APIs to fill in the blanks. For example, if your API consumer needs to retrieve a customer record using a `customerId` but only has the customer's `email` address, you'll need to design an API that accepts an email address and returns one or more customer records to choose from. Getting an understanding of the circumstances your API consumers are working with will help you design an API that allows them to solve their problems relatively quickly, easily, and—above all—safely.

With API First in mind, let's now take a look at the existing services we'll need to work with as we work to design and build an API for our client—BigCo, Inc.

• Click HERE to purchase this book now. discuss