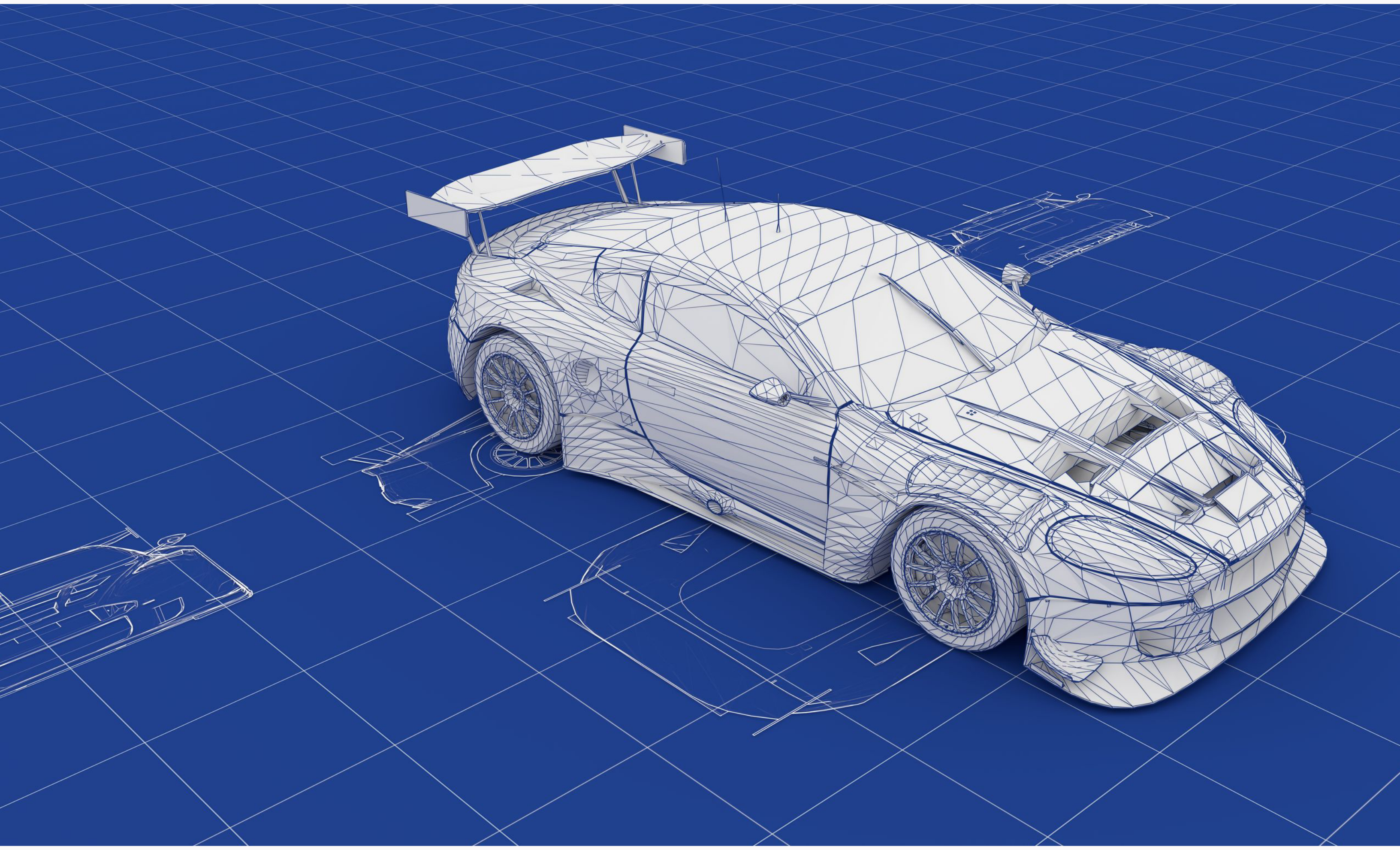
A Practical Approach to API Design

From Principles to Practice



D. Keith Casey Jr &
James Higginbotham

A Practical Approach to API Design

From Principles to Practice

D. Keith Casey Jr and James Higginbotham

This book is for sale at http://leanpub.com/restful-api-design

This version was published on 2019-01-23

Tweet This Book!

Please help D. Keith Casey Jr and James Higginbotham by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Just picked up the #APIDesignBook from @launchany and @caseysoftware. Looking forward to reading it!

The suggested hashtag for this book is #apidesign.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#apidesign

Contents

Welcome

There are only two hard things in Computer Science: cache invalidation and naming things.

– Phil Karlton[1]

Fundamentally, API design is hard. It combines two of the most challenging things in computer science and asks you to accomplish them in a distributed environment at scale. You might as well try it with your eyes closed, too.

In this book, we'll discuss the What, Why, When, and How behind RESTful APIs. We'll begin the story by describing what an API is. Then we'll discuss business and technical scenarios that might make an API make sense for your organization. Then we'll talk about how the core concepts fit together and the design principles that emerge. Throughout, we'll share practical real-world examples of all of the above. While many are success stories which can be emulated, many will be cautionary tales of strategies and implementations to avoid. Finally, while we'll touch on and explore the various religious aspects of API design, the goal of this book is to remain relatively neutral and give the different sides equal time.

While this is a book, we hope it also becomes a conversation. When you find something interesting, let us know. If you think we're wrong, say so. If you think we're right, tell your friends. If you want us to teach your team more and deeper concepts, please let us know.

D. Keith Casey, Jr. keith@caseysoftware.com
James Higginbotham james@launchany.com
February 2014

[1]http://martinfowler.com/bliki/TwoHardThings.html

Chapter 1 - APIs: An Introduction

What is an API

If you read the tech press, everyone knows they need an API but most aren't really sure what it is. They treat it as another checkbox like "Web 2.0" was a few years ago or a mobile app was most recently. In fact, there's an entire "API-first" movement in development circles that most people don't understand or even realize why. There are a variety of reasons for this situation with most reasons boiling down to a simple misunderstanding on what an API is and its purpose. Let's get the obligatory Wikipedia definition out of the way:

> *An application programming interface (API) specifies how some software components should interact with each other.*
>
> – Obligatory Wikipedia Definition[2]

An API doesn't have to be RESTful. It doesn't have to use SOAP. It doesn't have to use JSON or XML or OAuth or be built in Node or Rails or Python. It doesn't have to have pretty URLs or even necessarily use URLs at all. It doesn't have to make use of hypermedia concepts. The only requirement for an API to be an API is that it allows one program or software component to interact with another in a repeatable way.

It's almost disappointing, isn't it?

All of that said, an API *can* use all of those things listed above and definitely *should* use a number of them. When we speak about APIs in reference to web technology, it *usually* means a way of communicating over HTTP in a predictable and repeatable manner using payloads of XML or JSON to change the state of data within the system. More advanced systems use hypermedia and OAuth while less advanced or legacy APIs may use hard coded URLs and basic username and password-based authentication.

If that paragraph means nothing to you, don't worry about it. We'll get into it later. Many of the terms above are powerful concepts that help you build APIs in clean and predictable ways while also making it easy for your customers and users to adopt them. Because isn't that what matters?

There are two types of companies that have APIs. First are the type of companies that are primarily or entirely APIs. In this category, we find companies like SendGrid, Twilio, FullContact, and a host of others. In their case, the API *is* their business so it's absolutely necessary that their APIs are easy to use and quick to get started. The other category is businesses who have an API as one of their products. In this group we find Walgreens, Best Buy, FedEx, and many other traditional offline companies looking to expand their reach and presence online. It's worthwhile to remember that in many of these cases, they derive no direct revenue from API usage, which affect their strategy and implementation in a variety of ways.

[2]http://en.wikipedia.org/wiki/Application_programming_interface

At the end of the day, you must understand which category your organization will occupy. If usage of your API is directly connected to revenue, the goals are plain and simple: get people started quickly, make sure they keep using it, and give them reason to embed it throughout their organization. If an API is a checkbox that doesn't serve a business or technical case, it is doomed to failure. Luckily there are numerous business and technical cases your API could satisfy, regardless of your industry or position in the market.

Why you should have an API

If you believe the tech press, you should have an API because.. everyone should have an API. While this is great for the API management companies, it doesn't really apply to everyone. If an API doesn't provide tangible value to at least one major constituency within your organization, it is doomed to failure. Here are seven primary reasons on why an organization might need an API.

Business Cases

Mobile Strategy

The first business reason for an API is a mobile or mobile-first strategy. In most organizations, there are neither the time nor the resources to build tools for every single device. An API can ensure that additional - either less important or even unexpected - devices are addressed. In addition, it may parallelize the effort to create a presence much faster or differently than originally planned.

A real-world example is Netflix. When Netflix originally started mailing DVDs in 1997, its competition was Blockbuster and the cable providers. Over the next few years, Netflix quietly consumed the DVD rental market while they prepared to launch their streaming service in February 2007[3]. Seventeen months later in October 2008[4], they launched the official Netflix API with a simple goal and a strategy to get there by partnering with developers:

Netflix is everywhere

> *Millions of people love movies via Netflix, making this API an opportunity for all kinds of developers to add well-known value to any other application.*
>
> *The company says the API will allow access to data for 100,000 movie and TV episode titles on DVD as well as Netflix account access on a user's behalf.*

[3]http://en.wikipedia.org/wiki/Netflix#History
[4]http://readwrite.com/2008/09/30/netflix_api_launches_tomorrow

As a result, there were Netflix applications for most common devices within weeks or months. These initial applications provided customers with the ability to view and manipulate their queue whether they were online or offline. More importantly, it began to make Netflix a tiny - yet constant - part of everyday entertainment consumption. By the time Blockbuster entered the streaming market and later when they launched an API, Netflix was already the defacto standard. By that point, if a device had a full color screen, no matter its manufacturer, size, or profile, there was a Netflix application available that could enable watching a movie in just minutes.

Drive Usage

The next business reason for an API is to drive usage of a platform. Notice that this isn't necessarily *paid* usage but usage in general. Why is this scenario useful? By focusing on fast and simple usage of a platform, you can drive engagement and grow the user base even more.

Twitter used its API to drive usage

Twitter is a prime example. Their API was probably one of the most commonly cited for simplicity and ease of use. It was so easy that many coding frameworks changed their sample "getting started" project from "hello world" to "hello twitter." In a matter of months, there were literally hundreds of Twitter apps available on every device, ranging from desktops to browsers to more unique devices.

Twitter was not the first instant messaging or chat app and it has had to deal with competitors since launch. But by being ubiquitous on every device, it allowed people to step away from their computers and engage with the platform how and when they wanted. As a result, people used the platform more often than the alternatives. As people used the platform more, more people had to stay connected to it more often, further driving both passive (viewer) and active (tweeter) usage. This became evident at South By Southwest 2007[5] when a mobile-centric, desktop-disconnected audience found it as the primary means to trade information, keep in touch, and find out what others were doing.

Ubiquity may not have been their original goal for the API, but it created a virtuous cycle of usage driving viewing and viewing driving usage.

Defensive

The third business reason for an API is purely defensive. This is somewhat parsing words from the previous two scenarios but is distinct in a number of ways: when you're an established player in the market and competition disrupts your business, it can be a challenge to use your positioning, information, and infrastructure to make your existing business lines more stable. An API strategy may be able to launch you into new business lines and growth that was previously unrecognized.

The big box retailers have suffered the brunt of this sort of disruption. They have massive stores with massive inventories where they suffer from the "showrooming" problem. Potential customers come in, browse the merchandise, and once they find the item, they use a site like Amazon to find a

[5]http://mashable.com/2011/03/05/sxsw-launches/

better price and place the order. The customer walks out without the product but saved money and the retailer missed a potential sale.

Best Buy is working to turn that tide. By opening their API[6] and making massive amounts of data open to third parties, they have been able to build a community that is willing, and even excited, to explore the data and ways that it can be used. Some of the apps range from simple inventory explorers, but others include republishing the deals and discounts they are offering in store. *Yes, people are sharing ads on purpose.* To date, it has had a significant impact on their bottom line with a 15% increase in online sales in Q3 2013[7]. Note that this is before the majority of Christmas shopping days, such as Black Friday.

BestBuy Online API fights "showrooming"

Partner Connectivity

The final business reason is to improve connectivity with partners and suppliers. In the past, many organizations exchanged spreadsheets via email to update systems such as inventory management. If they were more advanced, the spreadsheets were uploaded via FTP and parsed automatically. Either way, these methods were error-prone and often fragile when changes would occur within either system. When importing a spreadsheet, a missing or new field, or even a misplaced comma, could cost hours of effort and delay expensive processes. Fundamentally, it was broken.

Alternatively, using APIs can provide a repeatable and consistent way to exchange information. The FedEx API demonstrates this from two different angles. First, by providing tight integration, an organization can simplify their own logistics by building on those within FedEx. On the other side, the organization's customers gain a great deal of insight into shipping, which was an opaque process just a few years ago.

FedEx specializes in partner connectivity

Technical Cases

Abstraction of Complexity

Powerful and important systems are almost always complex to use and integrate, often out of necessity. In the earliest days of the web, it took knowledge of the C programming language and

[6]https://bbyopen.com/developer
[7]https://www.internetretailer.com/2013/11/19/best-buys-online-sales-increase-151-q3

detailed understanding of SMTP in order to send an email. As these systems have become more important and widespread, they have become easier and less complex to use and integrate into our systems. This ease and simplicity comes from APIs.

At this stage, the code required to send an email is quite simple. Unfortunately, this simplicity doesn't include clickthrough tracking necessary for many emails. It also doesn't include configuring email servers that are resistant to both sending and receiving spam, delivery reports, open reports, and a variety of other aspects that are useful to you and your organization. Every single one of those features is additional code to write and maintain in the future.

SendGrid
Email Delivery. Simplified.

SendGrid simplifies sending & tracking email

On the other hand, SendGrid has a simple API for sending email. It only takes minutes to integrate into a system. The fundamental difference is that it does all of the above features - clickthrough tracking, spam resistance, delivery & open reports, etc - automatically. You don't have to do anything extra, just use the API. The API takes what is otherwise a complex system and allows you to make use of massive functionality in mere minutes with just a few lines of code.

Simplifying Interfaces

Despite numerous technical standards and long-established players, some systems are still terribly complex to interact with and use in general. One of the prime examples is your email inbox. Despite only two standards - POP3 and IMAP - there are so many implementations and variations of those implementations that what should be a handful of lines of simple code ends up being a deceptively complex implementation handling dozens of edge cases and unique scenarios. Building yet another email interface is a practice in futility likely to cause unforeseen schedule delays and unnecessary complexity. It's simply not worth the time or energy.

context.io

Context.io gives you an API for your inbox

In this scenario, an API can wrap all these variations and oddball cases and instead present a single unified interface for developers to use. The Context.io API does exactly this. To interact with an inbox - whether Microsoft Exchange, Gmail, or something else entirely - you initialize the inbox with your email credentials and then it transparently handles all the detail, complexity, odd scenarios, and other situations that occur. What could easily take days of effort is simplified to minutes. More importantly, it doesn't matter if the inbox is hosted in Microsoft Exchange, Gmail, or anything else, all are used exactly the same way with the exact same commands. The most powerful aspect is that the developer doesn't even have to know which email provider is being used. It's completely transparent.

Metered Usage

Many services lend themselves to simple monthly or flat rate pricing. For example, sending email has a marginal cost that is effectively zero so sending the hundred and first email costs nothing more than the hundredth. Alternatively, if we consider a resource such as CPU or processing time, the costs of running a server for two hours is effectively double the costs for one hour. In this case, a consumption-based or metered pricing model makes more sense. This is also known as "utility pricing" to represent that you are only billed for the portion of a resource used.

This is key for scenarios where massive amounts of processing time or power are required for short periods. A few years ago, the New York Times had just this situation[8] where they had 130 years of articles, approximately 11 million in total, which needed conversion into PDF documents. To solve the problem, they turned to Amazon EC2 to increase their processing power. As a result, the Times was able to temporarily utilize 100 servers to generate all of the required PDFs in under 24 hours. At the end of the project, the Times released those servers and was not charged any further.

All of the above

Of course, none of these scenarios are mutually exclusive. An API can be used to simplify interfaces, hide complexity, and still create a system for metered usage.

The Twilio API is an ideal example of addressing all of these challenges. Interacting with the telephone network is hard. There are numerous standards implemented at any given layer and each carrier implements those standards just a bit differently. To add to the complexity, carriers in different countries implement different standards. And to top it all off, the same carrier in different countries can use different standards to accomplish the same task. Further, interacting with all of these systems and components can be quite expensive. By combining all of these scenarios together, we can have a single API that abstracts away complexity and simplifies interfaces while limiting overall costs.

While the business cases for an API can be hard to quantify and therefore even harder to measure, the technical cases for an API are relatively simple and straightforward. Most developers understand building systems to make use of complex systems is a painful experience that they'd rather avoid. More importantly, they understand that replicating these systems in house often requires a huge amount of time and effort and is error prone. At the end of the day, building and using these APIs can save time, money, effort, and untold amounts of stress for all involved.

[8] http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/

Ready to learn more?

Thanks for reading our sample chapter. This book is a result of many years of designing and building web APIs and we hope that you learned a little more about them. The next step is to purchase the latest edition of the book, so that you can:

- Understand the principles of great API design
- Understanding REST vs. SOAP and how SOA and REST work together
- Strategies for building APIs into existing applications
- Build and integrate APIs for business workflows, not just data access
- Model business processes to drive the design of your APIs
- Design, document, and prototype your API
- Manage your API as a first-class product
- Build a community of developers that will adopt and share your API
- Implement common design patterns and idioms for APIs

Interested? Visit our publisher's page to purchase the latest edition of the book and save 10% today by clicking the following link: http://leanpub.com/restful-api-design/c/friends-save-10[9]

[9]http://leanpub.com/restful-api-design/c/friends-save-10

Want to stay informed about everything API-related?

The week in API strategy, news, articles, and upcoming events.

Email not displaying correctly? View it in your browser

A hand-curated weekly newsletter for API developers, sponsored by LaunchAny and CaseySoftware

Find this via Twitter? Subscribe now so you don't miss out

API Developer Weekly

June 5, 2014 - Issue #17

Hot Topics

How do APIs help build a business platform?
5 lessons on how to transform your business and industry into a platform that enables customers to build on top of your API. Companies discussed include WalmartLabs, Twilio, and Evernote.

Hello World Product API With Blueprint, RAML, And Swagger
A look at how to document your API definitions using three popular formats: Blueprint, RAML, and Swagger. A Hello World example is used to discuss and demonstrate the strengths of each format.

Swagger Creator Wants to Build Better SOAP for Web APIs
An article from a recent discussion with Tony Tam, the creator Swagger and CEO of Reverb on the Swagger 2.0 Working Group, recent news, and his views on completing web API definition languages.

Heroku's HTTP API Design Guide
A look at how Heroku built their internal HTTP platform API that drives everything both internally and for platform partners.

MQTT – Another Implementor's Perspective
Tim Kellogg looks at what can be improved with MQTT, a protocol commonly used for connecting devices, to make it better.

API Developer Weekly

While you wait, why not subscribe to the free API Developer Weekly? It is our hand-curated weekly email that is hyper-focused on the business, design, development, and deployment of APIs for web and mobile apps. It's great for developers, architects, product managers, and executives and serves as an easy way to stay up to date for free. View a sample newsletter and subscribe[10].

[10]http://bit.ly/api-dev-weekly-obs