Hypermedia In Action

1. Hypermedia

This chapter covers

- A reintroduction to the concept of hypermedia
- Why you might choose hypermedia over other approaches
- How hypermedia can be used to build modern web applications

Hypermedia is a universal technology today, nearly as common as electricity. Billions of people use a hypermedia-based systems every day, mainly by interacting with the *HyperText Markup Language (HTML)* over the *HyperText Transfer Protocol (HTTP)* via a Web Browser on the World Wide Web. People use these systems to get their news, check in on friends, buy things online, play games, send emails and so forth: the variety and sheer number of online services is truly astonishing.

And yet, despite this ubiquity, hypermedia itself is a strangely unexplored concept, left mainly to specialists. Yes, you can find a lot of tutorials on how to author HTML, create links and forms, etc. But it is rare to see a discussion of HTML *as a hypermedia*. This is in sharp contrast with the early web development era, when concepts like *Representational State Transfer (REST)* and *Hypermedia As The Engine of Application State (HATEOAS)* were constantly discussed and debated.

It is sad to say, but today HTML is, in some circles, viewed almost resentfully: it is considered a janky, legacy GUI description language that must be used build Javascript-based applications in, simply because that's what happens to be there, in the browser.

This as a shame, and we hope we can convince you that the hypermedia architecture is, instead, a tremendously innovative, flexible and *simple* way to build robust distributed systems. It deserves a seat at the table when you, the developer, are considering the architecture of your next online software system.

1.1. Why Hypermedia?

In this book we aim to re-introduce the reader to the concept of hypermedia, and show that it is a truly unique and powerful *network architecture*, to use the words of Roy Fielding. Fielding was an early developer of web technologies who gave us much of the language we use to discuss the World Wide Web's technical infrastructure.

Fielding recognized that the hypermedia architecture has a number of advantages over other network architectures, as evidenced by the fact that the web grew enormous so quickly: it is extremely simple compared to other approaches to building distributed applications, it survives network outages and changes relatively well and it is extremely tolerant of content and API changes. As someone interested in web development, these features all probably sound pretty appealing to you.

And, I assure you: they are appealing! You may reasonably be wondering: if hypermedia is so great,

why it has been abandoned by so many web developers today. There are, in our opinion, two core reasons: first, hypermedia *as hypermedia* hasn't advanced much since the late 1990s and early 2000s. HTML, the most widely used hypermedia, hasn't added any new ways to interact with a server with pure HTML in nearly two decades.

This somewhat baffling lack of progress has lead to the second reason: JavaScript and data-oriented JSON APIs took over web development as a way to provide more interactive web applications to end users.

It is unfortunate: this did not have to be the case. Rather than abandoning the hypermedia architecture, we could have kept pushing it forward and enabling more and more interactivity *within* that original, hypermedia model of the web. If we had done so, we could have retained much of the simplicity of the original web while still providing better user experiences.

And, in fact, there are some alternative front end libraries today that are attempting to do exactly this. Theese libraries use JavaScript not as a **replacement** for the hypermedia architecture, but rather use it to augment HTML itself *as a hypermedia*. One such library is htmx, was created by the authors and, in later chapters, we will show you how you can build many modern UX patterns for the web using the hypermedia model, often at a fraction of the complexity of more common, JavaScript-oriented solutions.

I this book helps you understand the fundamental REST-ful hypermedia architecture of the original web, and how you might use this architecture to build modern web applications. Even if you choose not to adopt hypermedia as a core technology for your own web development work (and it isn't an appropriate architecture for everything!) then at the very least you should come away with a deeper appreciation for this novel approach to building networked systems and understand where it might be applicable.

1.2. When should You Use Hypermedia?

> The emerging norm for web development is to build a React single-page application, with server rendering. The two key elements of this architecture are something like:
>
> 1. The main UI is built & updated in JavaScript using React or something similar.
> 2. The backend is an API that that application makes requests against.
>
> This idea has really swept the internet. It started with a few major popular websites and has crept into corners like marketing sites and blogs.
>
> — Tom MacWright, https://macwright.com/2020/05/10/spa-fatigue.html

Tom is correct: JavaScript-based Single Page Applications (SPAs) have taken the web development world by storm, offering a far more interactive and immersive experience than any old, gronky, web 1.0 HTML-based application could. Some SPAs are even able to rival native applications in their user experience and sophistication.

So, why on earth would you abandon this new, increasingly standard approach for an older, less popular one like hypermedia?

Perhaps you are building a web application that doesn't *need* a huge amount of user-experience innovation. These are very common and there is no shame in that! Perhaps your application adds its value on the server side, by coordinating users or by applying sophisticated data analysis. Perhaps your application adds value by simply fronting a well designed database with simple Create-Read-Update-Delete (CRUD) operations. Again, there is no shame in this!

In any of these later cases, using a hypermedia approach would likely be a great choice: the interactivity needs of these applications are not off the charts, and much of the value lives on the server side, rather on than on the client side. They are all amenable to what Roy Fielding calls "large-grain hypermedia data transfers".

By adopting the hypermedia approach for these applications, you will save yourself a huge amount of client-side complexity: there is no need for client-side routing, for managing a client side model, for hand-wiring in javascript logic. You will be able to focus your efforts on your server, where your application is actually adding value.

That does not mean, however, that your hypermedia application has to offer a relatively poor user experience! To the contrary, by using modern hypermedia tools like htmx and with occasional, judicious client side scripting, you can provide excellent user experiences while staying within the simple hypermedia model. For example, you may add an HTML input that filters the results showing in a table *as a user types into the input*, as Google does with search results. Believe it or not, this can be accomplished entirely with hypermedia, no heavy JavaScript required!

I think you will be quite surprised at just how sophisticated our hypermedia-based interfaces can get.

Now, that being said, there are cases where hypermedia is not the right choice. What would an example be?

One example that springs to mind is an online spreadsheet application, where updating one cell could have a large number of cascading changes that need to be made on every keystroke. Here we have a highly dependent user interface without clear boundaries as to what might need to be updated given a particular change. Additionally, introducing a server round-trip on every change would bog performance down terribly. This is simply not a situation amenable to "large-grain hypermedia data transfers" and we would heartly recommend a JavaScript-based approach to an application like this.

However, perhaps this spreadsheet application has a settings page, and perhaps that settings page *is* amenable to a hypermedia approach. If it is simply a set of relatively straight-forward forms that need to be persisted to the server, the chances are high that hypermedia would work great for this part of the app. And, by adopting hypermedia for that part of your application, you can save more of your complexity budget for the core, complicate spreadsheet logic.

What Is A Complexity Budget?

Any software project has a complexity budget, explicit or not: there is only so much complexity a given development team can tolerate and every new feature and implementation choice adds at least a bit more to the overall complexity of the system.

What is particularly nasty about complexity is that it appears to grow exponentially: one day you can keep the entire system in your head and understand the ramifications of a particular change, and a week later the whole system seems intractable. Even worse, efforts to help control complexity, such as introducing abstractions or infrastructure to manage the complexity, often end up making things even more complex. Truly, the job of the good software engineer is to keep complexity under control.

The surefire way to keep complexity down is also the hardest: say no. Pushing back on feature requests is an art and, if you can learn to do it well, making people feel like *they* said no, you will go far.

Sadly this is not always possible: some features will need to be built. At this point the question becomes: "what is the simplest thing that could possibly work?" Understanding the possibilities available in the hypermedia approach will give you another tool in that "simplest thing" tool chest.

1.3. OK, What Is Hypermedia?

> The English prefix "hyper-" comes from the Greek prefix "□περ-" and means "over" or "beyond"... It signifies the overcoming of the previous linear constraints of written text.
>
> — Wikipedia, https://en.wikipedia.org/wiki/Hypertext

Right. So what is hypermedia? Simply, it is a media, for example a text, that includes non-linear branching from one location to another, via, for example, hyperlinks embedded directly in the media.

You are probably more familiar with the term *hypertext*, from whose Wikipedia page the above quote is taken. Hypertext is a sub-set of hypermedia and much of this book is going to discuss how to build modern web applications with HTML, the HyperText Markup Language.

However, even when working with applications built mainly in HTML, there are nearly always other medias involved: images, videos and so forth, making *hypermedia* a more appropriate term for discussing applications built in this manner. We will use the term hypermedia for most of this book, to capture this more general concept.

1.4. HTML

> In the beginning was the hyperlink, and the hyperlink was with the web, and the hyperlink was the web. And it was good.
>
> — Rescuing REST From the API Winter, https://intercoolerjs.org/2016/01/18/rescuing-rest.html

Before we get into the more theoretical aspects of hypermedia, let's take a brief look at a concrete, familiar example of it: HTML.

HTML is the most widely used hypermedia in existence, and this book naturally assumes that the reader has a reasonable familiarity with it. You don't need to be an HTML or CSS ninja to understand the code in this book, but the better you understand the core tags and concepts of both HTML and HTTP, the more you will get out of this book.

Now, let's consider the two ur-elements of hypermedia in HTML: the anchor tag (which produces a hyperlink) and the form tag.

Here is a simple anchor tag:

Listing 1. 1. A Simple Hyperlink

```
<a href="https://www.manning.com/">
  Manning Books
</a>
```

In a typical browser, this tag would be interpreted to mean: "Show the text 'Manning Books' and, when the user clicks on that text, issue an HTTP GET to the url `https://www.manning.com/`. THen take the resulting HTML content from the response and use it to replace the entire screen in the browser."

This is the main mechanism we use to navigate around the web today, and it is a canonical example of a hypermedia link, or a hyperlink.

So far, so good. Now let's consider a simple form tag:

Listing 1. 2. A Simple Form

```
<form action="/signup" method="post">
  <input type="text" name="email" placeholder="Enter Email To Sign Up..."/>
  <button>Sign Up</button>
</form>
```

This bit of HTML would be interpreted by the browser roughly as: "Show an input and button to the user. When the user submits the form by clicking the button or hitting enter in the input, issue an HTTP POST to the relative URL '/signup'. Take the resulting HTML content in the response and use it to replace the entire screen in the browser."

I am omitting a few details and complications here: you also have the option of issuing an HTTP `GET` with forms, the result may *redirect* you to another URL and so on, but this is the crux of the form tag.

Here is a visual representation of these two hypermedia interactions:

Now, at this point, more experienced developers may be rolling their eyes. "I paid money to read *this*?"

But bear with me!

Consider the fact that the two above mechanisms are the *only* easy ways to interact with a server via HTML. That's barely anything at all! And yet, armed with only these two tools, the early web was able to grow exponentially and offer a staggeringly large amount of functionality to an even more staggeringly large number of people!

This is strong evidence of the power of hypermedia. Even today, in a web development world increasingly dominated by large javascript front end frameworks, many people choose to simply use vanilla HTML to achieve their goals and are perfectly happy with the results.

With just these two little tags, hypermedia manages to pack a heck of a punch!

1.5. So What *Isn't* Hypermedia?

Now let's consider another approach to interacting with a server:

Listing 1. 3. Javascript

```
<button onclick="fetch('/api/v1/contacts').then(response => response.json()).then(data
=> updateTable(data))">
    Fetch Contacts
</button>
```

Here we have a button element in HTML that executes some JavaScript when it is clicked. That JavaScript will issue a `GET` request to `/api/v1/contacts`. The response to this request will be in the JavaScript Object Notation (JSON) format. It is converted to a Javascript object and then handed off to the `updateTable()` method to update the UI based on the data that has been received.

This interaction is *not* using hypermedia. The JSON API being used here does not return a hypermedia response, it is rather a *Data API*, returning simple, plain old domain data. It is up to the browser, in its `updateTable()` method, to understand how to turn this plain old data into HTML, typically via some sort of client-side templating library.

This is a rudimentary single page application: we are not exchanging hypermedia with the server. Instead, we are, within a single page, exchanging *data* with the server and updating that page content.

Of course, today, the vast majority of web applications adopt more sophisticated frameworks for managing the user interface than this simple example: React, Angular, Vue.js, etc. With these more complex frameworks you typically work against a client-side model, updating JavaScript objects in memory and then allowing the UI to "react" to those changes via infrastructure baked into these modern libraries. Hence the term "Reactive" programming.

However, at the level of a network architecture, these more sophisticated are essentially equivalent

to the simple example above: they cast aside the hypermedia network model in favor of a data network model, exchanging JSON with the server.

1.6. Hypermedia Strikes Back

For many developers, since the rise of JavaScript and SPAs, hypermedia has become an afterthought, if it is thought of at all. You simply can't get the sort of modern interactivity out of HTML, the hypermedia we all use day to day, necessary for today's modern web applications.

But, what if history had worked out differently?

What if HTML, instead of stalling *as a hypermedia*, had continued to develop, adding new mechanisms for exchanging hypermedia with servers?

What if it was possible to build modern web applications within the original, hypermedia-oriented and REST-ful model that made the early web so powerful, so flexible, so... fun? Would hypermedia be a legitimate architecture to consider when developing a new web application?

The answer is yes, and there are a few libraries that are attempting to do exactly this: re-center hypermedia as a viable and, indeed, excellent choice for your next web application.

One such library is htmx, which the authors of this book work on, and which will be the focus of much of the remainder of the book. We hope to show you that you can, in fact, create many common "modern" UI features in a web application entirely within the hypermedia model and that, in fact, it is refreshingly simple to do so. And htmx is not alone: other libraries like unpoly.js and hotwire are working in this same conceptual space, making hypermedia, once again, the basis for building web applications.

In the web development world today there is a debate going on between SPAs and what are now being called "Multi-Page Applications" or MPAs. MPAs are, usually, just the old, traditional way of building web applications and thus are, by their nature, hypermedia oriented. Many web developers have become exasperated at the complexity of SPA applications and have looked longingly back at the simplicity and flexibility of MPAs.

Some thought leaders in web development, such as Rich Harris, creator of svelte.js, propose a mix of the two styles. Rich calls this approach to building web applications "Transitional", in that it attempts to mix both the old MPA approach and the newer SPA approach in a coherent whole.

We prefer a slightly different term to MPA. As we wish to emphasize the *hypermedia* aspect of the older (and, with htmx, newer) approach, we like the term *Hypermedia Driven Applications (HDAs)*. This clarifies that the core distinction between this approach and the SPA approach isn't the number of pages in the application, but rather the underlying *network* architecture.

What would the HDA equivalent of the JavaScript-based SPA-style button look like?

Done in htmx, it might look like this:

Listing 1. 4. an htmx implementation

```
<button hx-get="/contacts" hx-target="#contact-table">
    Fetch Contacts
</button>
```

As with the JavaScript example, we see that this button has been annotated with some attributes. However, in this case we do not have any imperative scripting going on. Instead we have *declarative* attributes, much like the `href` attribute on anchor tags and the `action` attribute on form tags. The `hx-get` attribute tells htmx: "When the user clicks this button, issue a `GET` request to `/contacts`". The `hx-target` attribute tells htmx: "When the response returns, take the resulting HTML and place it into the element with the id `contact-table`".

Note especially that the response here is expected to be in HTML format. This means that the htmx interaction is still firmly within the original hypermedia model of the web. Yes, htmx is adding functionality via JavaScript, but that functionality is *augmenting* HTML as a hypermedia, rather than throwing away hypermedia as the network model.

Despite being superficially very similar to one another, it turns out that this example and the JavaScript-based example it is based on are extremely different architectures. And, similarly, this approach is quite different than that taken by most SPA frameworks.

This may seem all well and good: a neat little demo of a simple tool that maybe makes HTML a bit more expressive. But surely this is just a toy. It can't scale up to large, complex modern web applications, can it? In fact, it can: just as the original web handled internet scale confoundingly well via hypermedia, due to its simplicity this approach can often scale extremely well with your application needs.

And, despite its simplicity, I think you will be surprised at just how much we can accomplish in creating modern, sophisticated user experiences in your web applications.

But before we get into the practical details of implementing a modern Hypermedia Driven Application, let's take a bit of time to make an in-depth study of the foundational concepts of hypermedia, and, in particular, of REST & HATEOAS, by reviewing the famous Chapter 5 of Roy Fielding's PhD dissertation on the web.