Hypermedia In Action

12. Other Hypermedia Libraries & Technologies

This chapter covers

- Hypermedia libraries beyond htmx and HyperView
- Non-HTML hypermedia technologies

Non-htmx Hypermedia Oriented Libraries

As the creators of htmx, we are obviously partial to its approach to building Hypermedia Driven Application. We like how it builds incrementally on top of existing concepts within HTML and stays "close to the metal" in that regard. However, it is far from the only library that takes hypermedia seriously as a technology. There are other excellent libraries that take different approaches, some similar some very different, while at the same time still respecting the core concept of the web: exchanging HTML with the server.

In addition to these HTML-based technologies, there is a whole world of non-HTML hypermedias. We have covered one in depth in this book: HyperView, a mobile hypermedia. But it is not the only non-HTML hypermedia out there, and it behooves us to look at some of these other technologies as well.

12.1. Unpoly

Unpoly is a mature and stable hypermedia-oriented library that has been in existence for nearly a decade. It came out of the Rails community in the 2010s, which was a hot-bed for innovation in the web application world at that time.

Unpoly provides quite a bit more functionality than htmx does. For example, it supports:

- A wide array of animations for transitions, such as sliding
- A notion of "layers", allowing for native modal-like behavior
- Built in preloading of content (in htmx this requires an extension)
- A strong focus on progressive enhancement, so non-JS enabled clients continue to work

Like htmx, Unpoly uses attributes on elements to add enhancements beyond their normal

behavior. However, unlike htmx, Unpoly strongly encourages the use of standard anchors and forms for hypermedia interactions. This is in keeping with its focus on graceful degredation when JavaScript is not available.

Here is some sample Unpoly code, to give you a flavor of the library:

Listing 12. 1. An Unpoly Powered Link

```
<a href="/contacts" up-follow
                    up-target=".contacts-table">
  Get Contacts
</a>
```

Here you can see the `up-follow` attribute, which tells Unpoly "when a user clicks this link, issue the request as an AJAX request instead". You can also see the `up-target` attribute, which, like the `hx-target` attribute in htmx, allows you to specify the target element for replacement.

Like the `hx-boost` attribute we looked at earlier, Unpoly will update the navigation bar with the new URL, updating history.

12.1.1. Layers

Unpoly has a concept of layers, which allows you to open new content in an overlay. This allows for a more sophisticated notion of windowing in your web application that is available via the normal DOM API. Layers can be shown as modal dialogs, as a drawer that slides out from the side, as a popup anchored to a given link or as an overlay, in addition to the normal "root" layer, where content will replace the current window.

As an example, here is a screenshot from the Unpoly demo application, available at https://demo.unpoly.com

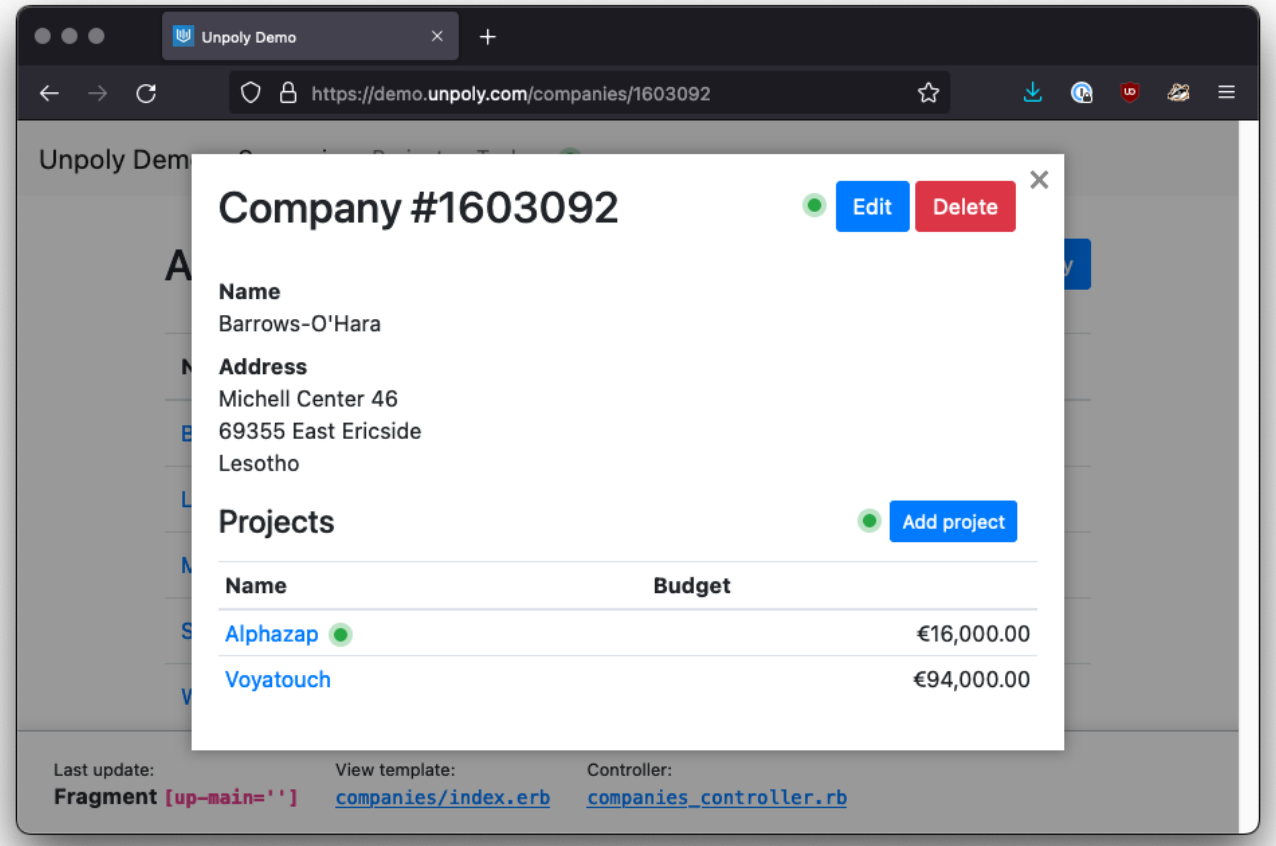

Figure 12. 1. An Unpoly Modal

This modal was launched by clicking on a normal anchor tag link that had been annotated to indicate that the result should be placed in a new modal layer. Unpoly provided the infrastructure to show the response from the server within a modal window, rather than replacing the entire screen, as well as dismiss the window. And a really nice feature here is that, if the user does not have JavaScript enabled, the web application continues to work, since the link will simply follow the normal "replace the entire screen" logic of web applications.

12.1.2. Unpoly's JavaScript API

Unpoly ensures that all functionality that is available via attributes is also available via JavaScript, allowing complete access to the underlying concepts and ideas. This makes Unpoly a great library to pair with JavaScript scripting in your Hypermedia Driven Application.

Given the link above to retrieve `/contacts`, you can write the following code:

Listing 12. 2. Triggering An Unpoly Powered Link Via JavaScript

```
var link = document.querySelector('a')
up.follow(link)
```

And Unpoly will load the link just like if the user had clicked on it.

Unpoly is a great option to consider when building a Hypermedia Driven Application. It provides more structure than htmx does, and focuses on progressive enhancement. If these factors are important to you, Unpoly might be a better choice that htmx. htmx, on the other hand, is lower level and stays closer to HTML conceptually.

12.2. Hotwire

Hotwire is a collection of libraries produced by 37 Signals, the company that also created the Ruby on Rails web framework. Hotwire stands for "HTML Over The Wire", and consists of three different components:

- Turbo - A library for building HTML-driven web applications
- Stimulus - A library for writing JavaScript within a Turbo-based web application
- Strada - A mobile library for integrating with HTML-drive web applications

Turbo is the core library of Hotwire, and, like Unpoly, it offers a lot more structure than htmx does. Turbo has the concept of "turbo frames", which are pre-defined areas that can be updated independently from one another. To take the Unpoly example above and convert it to Turbo, we would remove the direct annotations on the anchor tag and instead wrap it in a `turbo-frame` element, which is a custom HTML element provided by the Turbo library:

Listing 12. 3. An Turbo Powered Link

```
<turbo-frame id="contacts-table">
     ...
    <a href="/contacts">
      Get Contacts
    </a>
    ...
</turbo-frame>
```

When a user clicks on this link, the returned content will be inserted into the turbo-frame

appropriately.

12.2.1. Web Sockets

A unique feature of Turbo is that it can use Web Sockets for exchanging hypermedia with a server. Web Sockets are a browser technology that allows for a persistent connection to be maintained with a server. By maintaining a persistent connection, you can avoid the set up and tear down costs associated with establishing an HTTP connection, and you also allow the server to "push" messages down to the client.

Turbo supports a `turbo-stream` element that can be tied to a Web Socket, allowing you to stream HTML down to the client. This can make for a very dynamic user interface and works well for things like an online messaging application. (Hotwire was designed in large part to support the Hey! online email application, an email client that has many instant messaging-like features.)

Hotwire would be a particularly good choice for your Hypermedia Driven Application if you are using Ruby on Rails as your back end server: since both technologies are developed by the same people, they work seamlessly with one another. Hotwire can be used with other back end technologies, as well. As with Unpoly, Hotwire gives you more structure than htmx does, with a notion of "frames" and "streams". Hotwire also supports streaming HTML over Web Sockets, something that htmx supports only secondarily.

Hotwire also has the advantage that it is supported by a mature and famous company, who depends on it for their own applications.

12.3. jQuery

You may be surprised to see jQuery on this list. jQuery is the granddaddy of them all of JavaScript libraries, and is where many older developers first started exploring AJAX, using the `ajax()` function. jQuery is typically referred to by the somewhat cryptic `$` symbol.

Let’s take the example "Get Contacts" link that we have been using and convert it to jQuery. To do so we are going to have to get a bit into the weeds with respect to how jQuery works.

When you are writing jQuery code, you tend to separate out your JavaScript from your HTML, a concept we discussed in Chapter 9 called "Separation of Concerns". The general

pattern is: in the "ready" event for the page (that is, when the page is fully rendered), look up elements in the page and wire in event handlers.

In our case, we are going to look up the "Get Contacts" element, which we will change to a button, since we don't want the link behavior anymore. We do this using the jQuery query syntax, which is to pass a CSS selector into the `$()` function. Don't worry too much about the details if you aren't familiar with jQuery, the important thing to understand conceptually is that we "look up" the button and then wire in a click handler.

From the click handler, we can invoke the `ajax()` method in jQuery. We can then take the result and put it into another element by looking that element up and calling the `html()` function with the HTML content that is return by the server.

Here is what our code might look like:

Listing 12. 4. Issuing an AJAX Request in jQuery

```
<script>
    $(document).ready(function() {
        $("#contacts-btn").click(function() {
            $.ajax({url: "/contacts",
                    success: function(result) {
                $(".contacts-table").html(result);
            }});
        });
    });
</script>
<button id="contacts-btn">
  Get Contacts
</button>
```

Not the cleanest looking code in the world, at least to our eyes, but it works, and it is using hypermedia: we are not exchanging JSON with the server here. We are, instead, pulling down HTML and placing it in the DOM, just like htmx does.

This style of jQuery code, in fact, was the early inspiration of intercooler.js, the predecessor to htmx. intercooler.js was built on top of jQuery, initially as a jQuery extension and then as a stand-alone library. htmx was created as "intercooler 2.0" and the jQuery dependency was removed.

12.4. VanillaJS

Although still widely used, jQuery has become less popular over time as JavaScript has standardized across browsers, and as the native JavaScript APIs have improved. (These improvements have often been inspired by jQuery!) Today many people would prefer to use plain JavaScript.

It turns out that using plain JavaScript to implement a simple Hypermedia Driven Application is pretty easy today, thanks to the addition of the `fetch()` API! The `fetch()` API allows you to issue AJAX requests in a much simpler manner than the older `XMLHttpRequest` API: you can simply call a function, `fetch()`, with the URL that you want to issue the HTTP request to.

Let's rework the jQuery button above to use `fetch()`. We will inline the code directly on the button using an `onclick` handler, which is more in line with how people use JavaScript today:

Listing 12. 5. Issuing an AJAX Request using `fetch()`

```
<button onclick="fetch('/contacts')
                  .then((response) => response.text())
                  .then((data) =>{
                      document.querySelector('.contacts-table').innerHTML = data;
                  };">
  Get Contacts
</button>
```

As with the jQuery example, not exactly easy on the eyes, at least in our opinion, but it works! We call the `fetch()` function, passing in the path that we want to issue the AJAX request to, then convert the response to text, then look up the contact table by a class name, and then set its inner HTML to the content that came back from the server.

I hope you can see why we were inspired to create htmx, based on this code: there is an awful lot of syntax to achieve a relatively simple end goal of pulling down some HTML and loading it into the DOM. Nonetheless, if you only need to do this style of AJAX request in a few spots in your web application, the fact that you don't need any additional libraries using this approach may outweigh the syntactic convenience and additional functionality that htmx provides.

12.5. Non-HTML Hypermedia

We have focused on two hypermedia formats in this book: HTML and HyperView. These two hypermedia formats address the two most common platforms for building online applications: the browser and mobile clients. HTML is the most widely known and deployed hypermedia in the world, whereas HyperView is a relatively new and unknown technology. But are there other hypermedia formats out there?

It turns out that yes, there are.

12.5.1. Atom

Atom is an XML-based hypermedia format that support publishing and editing web resources. It is designed to help websites publish a feed of updated content in a standard hypermedia format. It was developed as an alternative to the Real Simple Syndication (RSS) format, an updated format that resolved some of the problems with that earlier format.

After HTML, Atom is probably the most widely deployed hypermedia format. It became very popular for news and blog-style websites in the late 2000s.

Atom explicitly includes hypermedia controls, the ability to edit resources and follow urls, via the `link` element, which can include a `rel` attribute (short for "relation") indicating exactly what the links relationship to the given representation is.

Here is an example Atom document:

Listing 12. 6. An Example Atom Document

```
<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
    <title>Atom-Powered Robots Run Amok</title>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <author><name>John Doe</name></author>
    <content>Some text.</content>
    <link rel="edit" href="http://example.org/edit/first-post.atom"/> ❶
</entry>
```

❶ A hypermedia control, telling an atom client where this entry can be updated

As you can see, Atom looks, to an extent, like HTML, but HTML tailored to represent a feed of articles. The `link` element within the document is a hypermedia control, telling an Atom client where this resource can be updated via either a `PUT` or `DELETE`.

The Atom hypermedia format might be useful for you if you are, for example, creating a CMS or blogging system, and wish to expose a feed using a hypermedia format.

12.5.2. Hypertext Application Language (HAL)

The Hypertext Application Language (HAL) is another non-HTML hypermedia. Interestingly, this hypermedia supports *both* XML and JSON. Since most of the JSON APIs we have been looking at in this book have been Data APIs, let's take a look at how HAL imposes a hypermedia on top of that format.

In its JSON form, HAL consists of a root JSON object with two reserved "meta-properties":

- `_links` - contains links to other resources
- `_embedded` - contains embedded resources

The `_links` property in JSON responses provides hypermedia control information for the current resource. For a collection of contacts, here is what a `_links` property might look like in a HAL response:

Listing 12. 7. A HAL `_links` property

```
"_links": {
  "self": { "href": "/contacts" },
  "next": { "href": "/contacts?page=2" },
  "find": { "href": "/contacts{?id}", "templated": true }
```

The `_embedded` property is used to enclose other resources, such as a collection of contacts with their own HAL controls. In the case of our collection of contacts, this would include the details of each contact, as well as nested `_links` and/or `_embedded` entries within them.

If we were to add HAL support to the contacts Data API we built in Chapter 9, we woudl add the *`links`* `property as a top level property in our response, and move the contacts property inside of the embedded` property. Our response might look something like this:

Listing 12. 8. Updating our Contacts Data API to include HAL controls

```
{
  "_links": {
    "self": {
      "href": "/contacts"
    },
    "next": {
      "href": "/contacts?page=2"
    },
    "find": {
      "href": "/contacts{?id}",
      "templated": true
    },
    "_embedded": {
      "contacts": [
        {
          "_links": {
            "self": {
              "href": "/contacts/2"
            }
          },
          "email": "carson@example.comz",
          "errors": {},
          "first": "Carson",
          "id": 2,
          "last": "Gross",
          "phone": "123-456-7890"
        }
      ]
    }
  }
}
```

Our JSON API now includes hypermedia controls! The top level `_links` property includes links to both the current and next page of contacts. And, within the `_embedded` property, you can see nested `_links` properties for the contacts, with a "self" link to the URL for the given, embedded contact. Very cool!

This approach to building a JSON API exposes a general, uniform API to the outside world, and a proper hypermedia client will be able to work with these resources without any deep understanding of the internal meaning of the data.

12.5.3. Creating Your Own Hypermedia Format

Another approach, one championed by Mark Amundsen, is to build your *own* hypermedia. This may sound a bit crazy at first, but Mark has written a series of books on exactly this topic. One, in particular, that we can recommend is "Restful Web Clients", which goes over the design of both good hypermedia APIs as well as how to write good hypermedia **clients** for those APIs.

In our experience, the idea of hypermedia APIs are fairly clear to most developers. However, understanding how to create a proper hypermedia **client** is a tricky, and much less discussed part of the hypermedia puzzle. One of the reasons we are so excited about HyperView, and were thrilled to have Adam Stepinski, the creator of HyperView, join us as a co-author of this book, is because he did the hard work of not only defining a hypermedia format, but also of creating a hypermedia client that can work with that format. By creating both sides of the hypermedia puzzle for mobile application, Adam has made HyperView a far more practical and useful technology!

12.5.4. Summary

- In this book we've looked at two hypermedia-based technologies in depth: htmx and HyperView
- There are other hypermedia-oriented libraries out there worth considering, especially for web development. Unpoly and Hotwire are two popular ones.
- There are hypermedia beyond HTML, such as Atom, a hypermedia for representing feeds of articles and HAL, as simple hypermedia format for enhancing your JSON APIs with hypermedia controls.
- If you are ambitious, you might even consider creating your own hypermedia format and client!