The Pragmatic Programmers

Facets of Ruby

Series editor: Noel Rappin

Hotwire Native for Rails Developers

Build Native Mobile Apps Using Your Server

Joe Masilotti

edited by Nicole Taché

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Copyright © The Pragmatic Programmers, LLC.

CHAPTER 2

Control Your Apps with Rails

The biggest benefit of using Hotwire Native to build mobile apps is that you're rendering content directly from the Rails server. Make a change to your Rails code and boom - the iOS and Android apps get that update automatically. But it can be hard to appreciate that magic until you see it for yourself.

This chapter covers different techniques and approaches to drive content and behavior in the apps directly from the server. Everything in this chapter will be Ruby on Rails code.

Specifically, you'll learn how to:

1. Dynamically set a native title via HTML
2. Hide the navigation bar with conditional Ruby
3. Hide the navigation bar with CSS
4. Add back in missing elements
5. Keep users signed in between app launches
6. Access the User's Camera and Photos on iOS
7. Access the User's Camera and Photos on Android

But before we explore each technique, I want to show you something - probably the most exciting part about building Hotwire Native apps.

First, visit `https://localhost:3000` in your browser and open Xcode to run the iOS app. Both show the hike index page like expected.

Now add a new HTML element to the code that powers this page, like so:

ch02_01/rails/app/views/hikes/index.html.erb

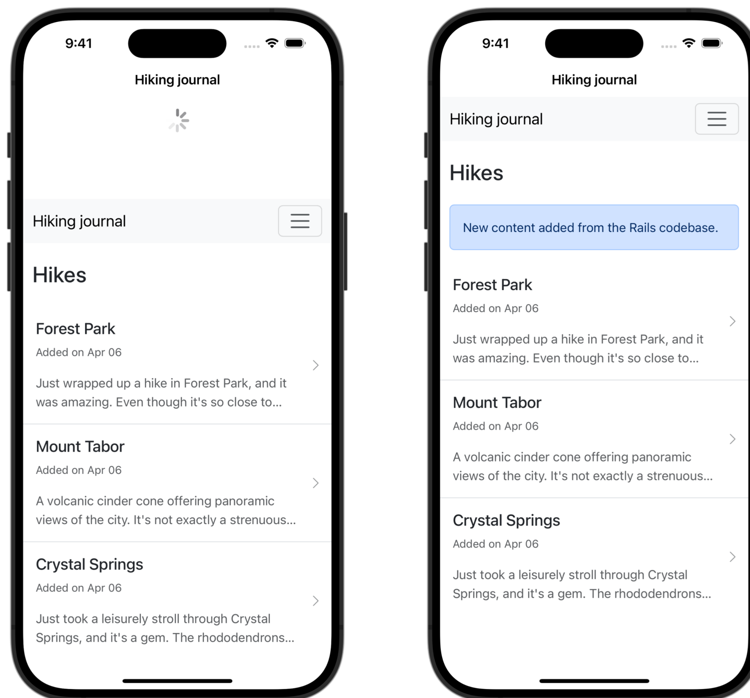
```
<%= render "shared/header", title: "Hikes" %>

➤ <div class="container">
➤   <div class="alert alert-primary" role="alert">
➤     New content added from the Rails codebase.
➤   </div>
```

```
</div>
<%# ... %>
```

Refresh the page on the browser to see the updated content. Boring, right? That's what's expected of websites! But now for the magic…

Back in the iOS simulator, click and drag the mouse from the top of the screen down toward the bottom. You'll see a little spinner start filling in at the top. Once it fills, let go and the page will refresh. And there's the content you added to the server!



The magic is that this didn't require any native code changes, a rebuild of the app, *or* a deploy to the App Store. Whatever content your Rails server renders, your apps will always show on the screen.

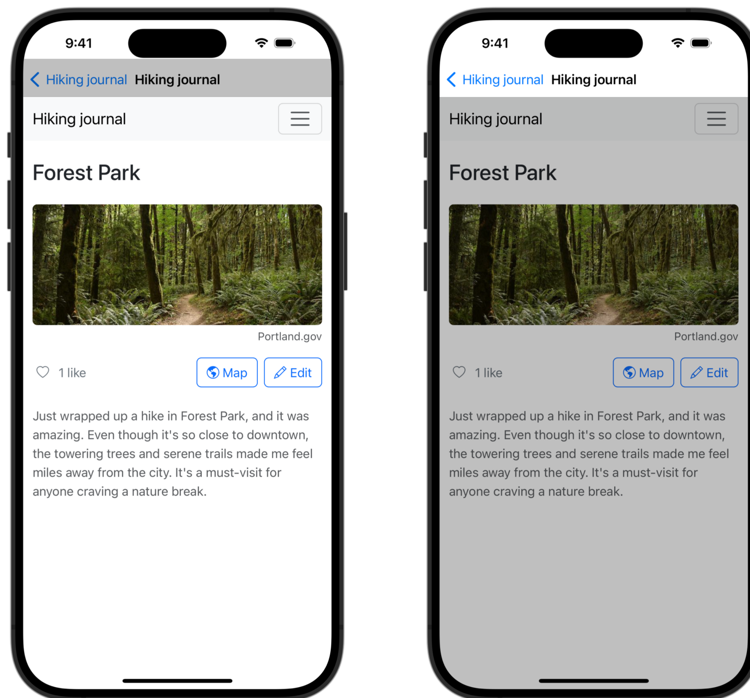
I'm using an alert to keep things simple. But the real magic comes from the mindset shift - you could add an entire new feature to the Rails codebase and the iOS and Android apps will get them…entirely for free.

Ready to learn the first technique? Let's go!

Dynamically Set a Native Title via HTML

We'll cover the first technique in our list – dynamically setting the native title via HTML – by way of addressing the repetitive "Hiking Journal" text at the top center of the screen. This *title* is most similar to an `<h1>` tag on the web, a string that represents the content on the screen. Right now, every screen shows the same text at the top. It would look a lot nicer, and would actually be more useful, if that title displayed something a bit more dynamic.

The title lives inside the native navigation bar, highlighted in the left image that follows. The web content rendered from the server is highlighted on the right.



Under the hood, Hotwire Native sets this title element to the HTML's <title> tag when the page loads. And it does so automatically! This means we can update the string directly from our HTML code. No additional Swift or Kotlin required.

But how do we set a different <title> for each page? I like to use a Rails helper: `content_for`.

content_for

content_for[1] allows you to insert content into named blocks in your layout. Think of it as a placeholder for content created later in the page rendering cycle.

You'll interact with content_for in two ways: to set content and to retrieve content. You set content by providing two parameters: the identifier of the content and the content itself. Then, you retrieve it by only passing in the identifier, as follows:

```
<%# Set content. %>
<% content_for :name, "Joe" %>

<%# Retrieve (and render) the content. %>
<%= content_for :name %>
```

You can also pass a block to set more complex content, like HTML:

```
<%# Set content from a block. %>
<% content_for :complex_html do %>
  <%= link_to "Home", root_path %>
<% end %>

<%# Render the block content. %>
<%= content_for :complex_html %>
```

If no content has been set for an identifier, then content_for returns nil. This can be used to provide a default value to fallback to by chaining with ||.

```
<%= content_for(:name) || "Someone" %>
```

Setting Dynamic Titles

Using content_for, let's set the HTML title of the hike page to the name of the hike. This will, in turn, set the native title of the view controller in Hotwire Native.

First, refactor the application.html.erb by replacing the contents of the <title> tag with content_for. Don't forget a default value!

ch02_02/rails/app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
➤   <title><%= content_for(:title) || "Hiking journal" %></title>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%# ... %>
```

1. https://guides.rubyonrails.org/layouts_and_rendering.html#using-the-content-for-method

```
  </head>

  <body>
    <%# ... %>
  </body>
</html>
```

The show template renders a small partial at the top of the file: shared/_header.html.erb.

ch02_03/rails/app/views/hikes/show.html.erb

```
➤ <%= render "shared/header", title: @hike.name %>

<div class="container">
  <%# ... %>

</div>
```

Open that partial, app/views/shared/_header.html.erb, and use content_for to set the title of the page to the name of the hike via the title local variable.

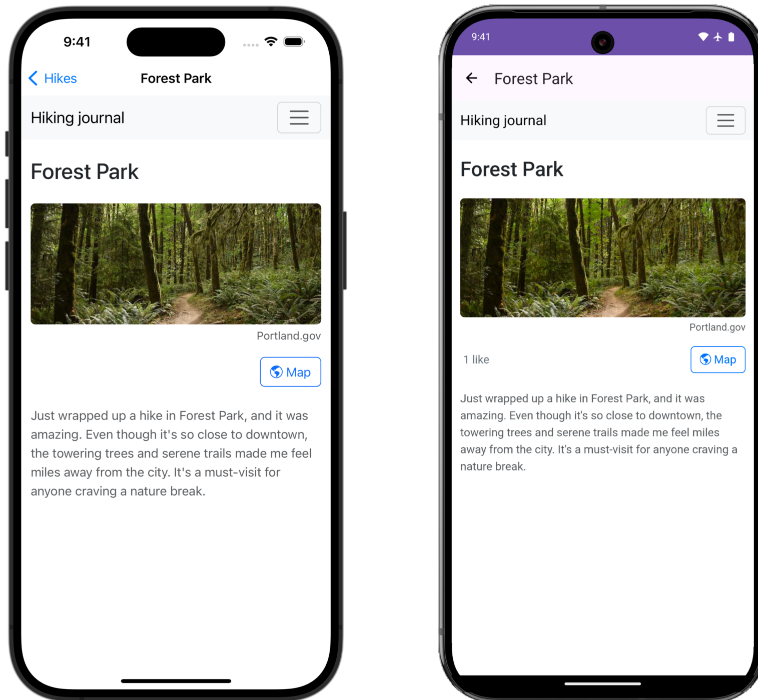
ch02_04/rails/app/views/shared/_header.html.erb

```
➤ <% content_for :title, title %>

<div class="container">
  <h1 class="my-4 pt-md-4"><%= title %></h1>
</div>
```

Run the app and navigate to a hike page. The title of the hike now appears as the native title of the view controller. And because you used content_for to set it dynamically, every single hike page gets this functionality.

Here's what that looks like on iOS (on the left) and Android (on the right).



Hide the Navigation Bar with Conditional Ruby

Up next is addressing the web-based navigation bar that appears on every page. On mobile web, this navigation bar is a convenient way to display the title of the website and a few important links - like a hamburger menu of sorts. But your native apps have their own navigation bar. Native ones! There's no need to display the same content twice in the apps. In this section, you'll hide specific web elements for the Hotwire Native apps.

First, we need a way to identify the Hotwire Native app from other web requests. Lucky for us, the turbo-rails gem[2] comes with a helper to do exactly that. The Rails app is already using the gem, so no changes are needed on your end.

For the curious, here's what the code looks like under the hood of the gem:

```
# turbo-rails/app/controllers/turbo/native/navigation.rb

module Turbo::Native::Navigation
  extend ActiveSupport::Concern

  included do
    helper_method :hotwire_native_app?, :turbo_native_app?
```

2. https://github.com/hotwired/turbo-rails

```
  end

  def hotwire_native_app?
    request.user_agent.to_s.match?(/(Turbo|Hotwire) Native/)
  end

  # ...
end
```

This exposes the helper method hotwire_native_app? to our Rails controllers and views in order to easily identify requests from Hotwire Native apps. If the user agent includes the “Hotwire Native” string, then the method returns true. Hotwire Native automatically adds “Hotwire Native” (and “Turbo Native”, for backwards compatibility with the old library) to the user agent on every request, so no additional code is needed in the app.

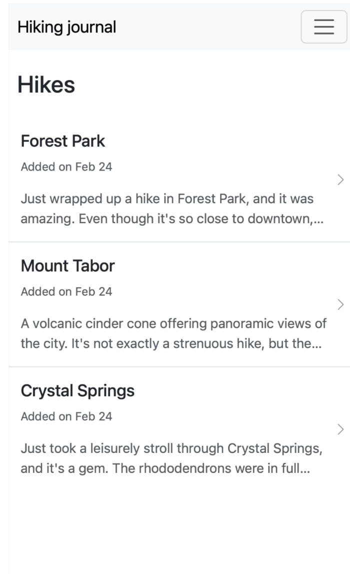
Use this method in app/views/shared/_navbar.html.erb to skip the rendering of the navigation bar for Hotwire Native apps, as follows:

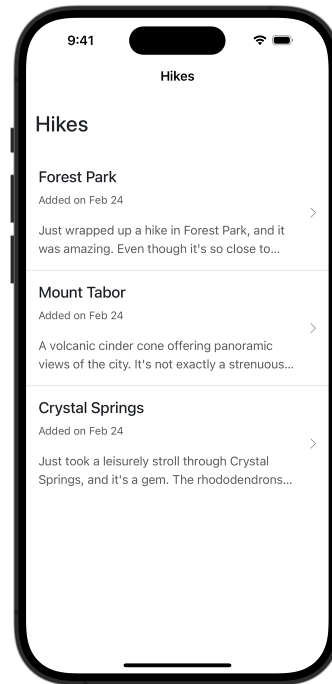
ch02_05/rails/app/views/shared/_navbar.html.erb

```
➤ <% unless hotwire_native_app? %>
    <nav class="navbar bg-body-tertiary">
      <div class="container">
        <%# ... %>
      </div>
    </nav>
➤ <% end %>
```

The navigation bar is now hidden from the native apps but is still visible on mobile web.

• Click HERE to purchase this book now. discuss





Using `hotwire_native_app?` to conditionally render content works great. You are given full control over what to send to the native apps and what to only render on mobile web. But this approach has a trade-off. Sending different HTML, depending on the user agent, can break caching. To know which HTML to send, the cache now *also* needs to be aware of the user agent of the request.

Personally, I'd rather not double the size of my cache just to hide elements in the apps. Let's take a look at a different approach to hiding the navigation bar, which sends the same HTML no matter the user agent - no changes to your caching required.