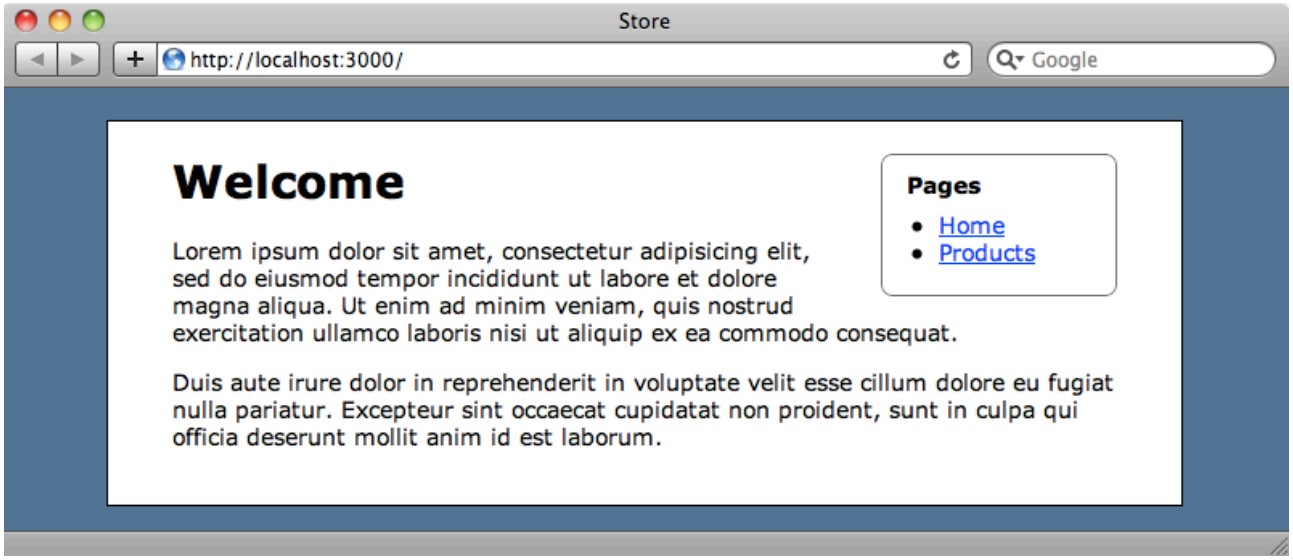
Episode 269

Template Inheritance

One of the new features in Rails 3.1 is template inheritance. This feature won't revolutionize the way you write your views, but it's a useful new feature to have and in this episode we'll show you how it works.

For the purposes of this episode we have an application with the same navigation on each page.



The navigation is currently defined in the application layout file within a `div` element with an `id` of `side` and is static.

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

```
<!DOCTYPE html>
<html>
<head>
  <title>Store</title>
  <%= stylesheet_link_tag "application" %>
  <%= javascript_include_tag "application" %>
  <%= csrf_meta_tags %>
</head>
<body>
  <div id="container">

    <% flash.each do |name, msg| %>
      <%= content_tag :div, msg, :id => "flash_#{name}" %>
    <% end %>

    <div id="side">
      <strong>Pages</strong>
      <ul>
        <li><%= link_to "Home", root_path %></li>
        <li><%= link_to "Products", products_path %></li>
      </ul>
    </div>

    <%= yield %>

    <div class="clear"></div>
  </div>
</body>
</html>
```

We want to customize the navigation so that it changes depending on the current controller. There are a number of ways that we can do this but we're going to use template inheritance. To do this we'll first move the navigation out into a partial that called "side".

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

```
<div id="side">
  <%= render "side" %>
</div>
```

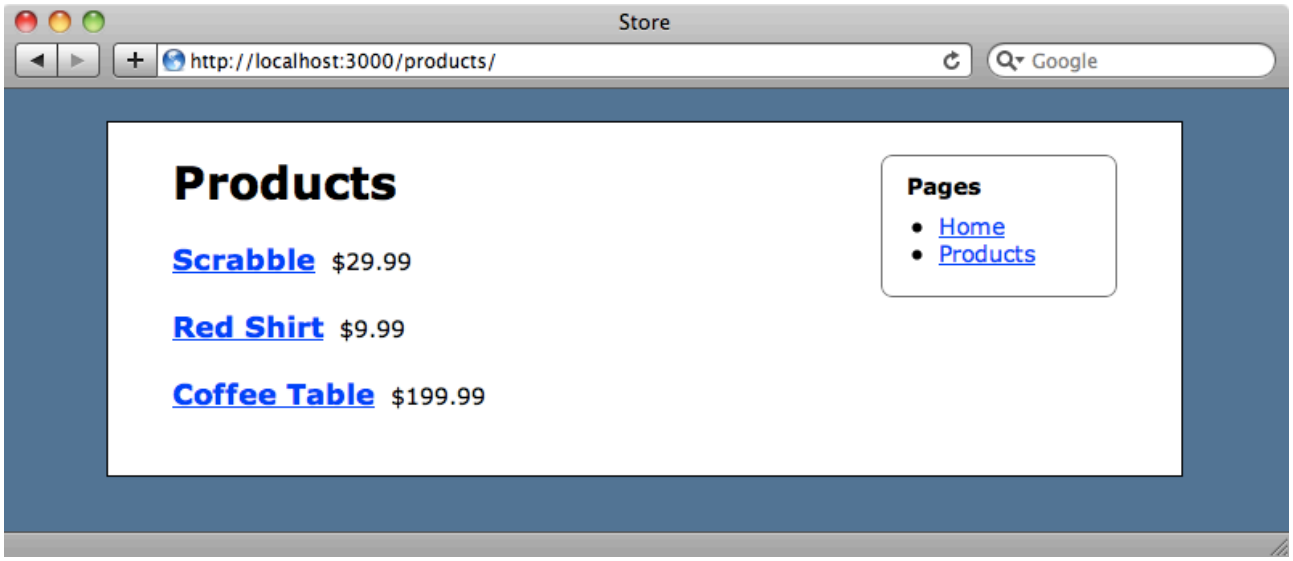
Where should we put this new partial so that it's accessible from every controller in our application? We could create a directory called `shared` under the `views` directory and put it there or we could place it under the `layouts` directory but in Rails 3.1 we can put it in an `application` directory under the `views` directory. This directory doesn't exist by default in a Rails 3.1 application so we'll have to create it.

`/app/views/application/_side.html.erb`

```
<strong>Pages</strong>
<ul>
  <li><%= link_to "Home", root_path %></li>
  <li><%= link_to "Products", products_path %></li>
</ul>
```

This `side` partial is now available to all of the controllers as the view inheritance works in parallel with the controller inheritance. All controllers inherit from `ApplicationController` so all templates will be inherited from the `application` directory.

If we load one of our application's pages now the navigation works as before.

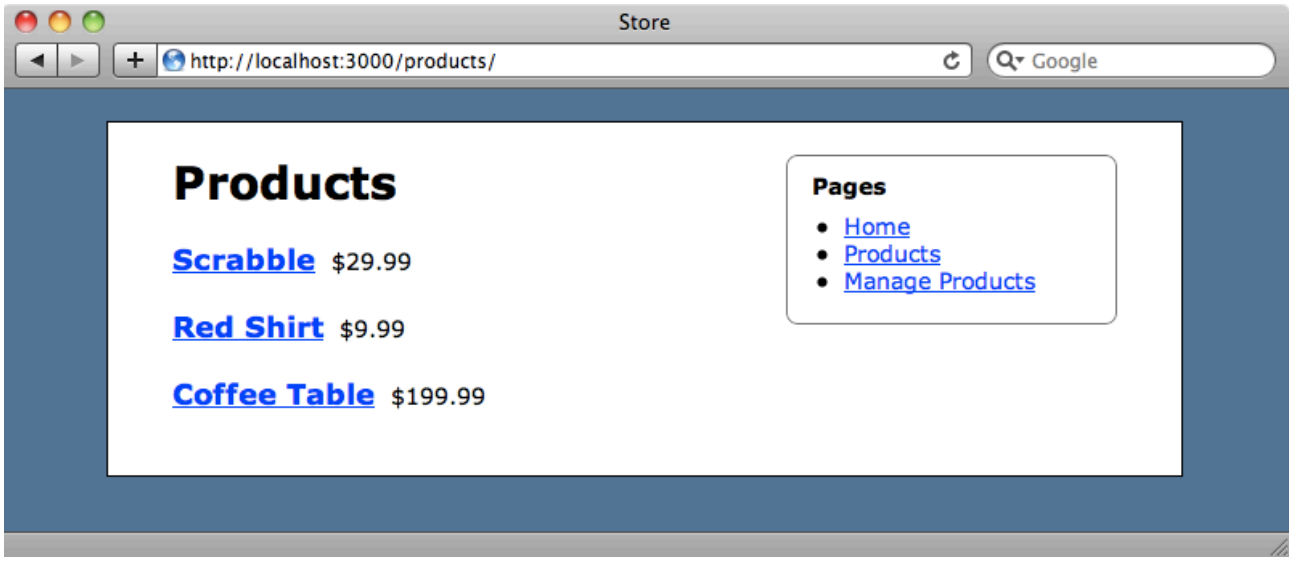


We now have a partial that we can easily override in any of our application's controllers. All we have to do is create a `_side.html.erb` partial file in another controller's `views` directory. We'll do this now for `ProductsController`, creating a partial here that has an additional link.

/app/views/products/_side.html.erb

```
<strong>Pages</strong>
<ul>
  <li><%= link_to "Home", root_path %></li>
  <li><%= link_to "Products", products_path %></li>
  <li><%= link_to "Manage Products", admin_products_path %></li>
</ul>
```

If we reload any page under the `ProductsController` now we'll see this new link. Viewing a page under any other controller will still show us the default navigation.



We now have a standard location where we can place templates that will be shared across controllers and we can override any of these templates by creating a template with the same name in the appropriate controller's `views` directory.

This approach works across deep-nested controllers too. Our application has several controllers under an `Admin` namespace, all of which inherit from `BaseController`.

/app/controllers/admin/base_controller.rb

```
module Admin
  class BaseController < ApplicationController
  end
end
```

We can override templates in this `Base` view, too. If we click the "Manage Products" link in the page above we'll be taken to the `/admin/products` page and the

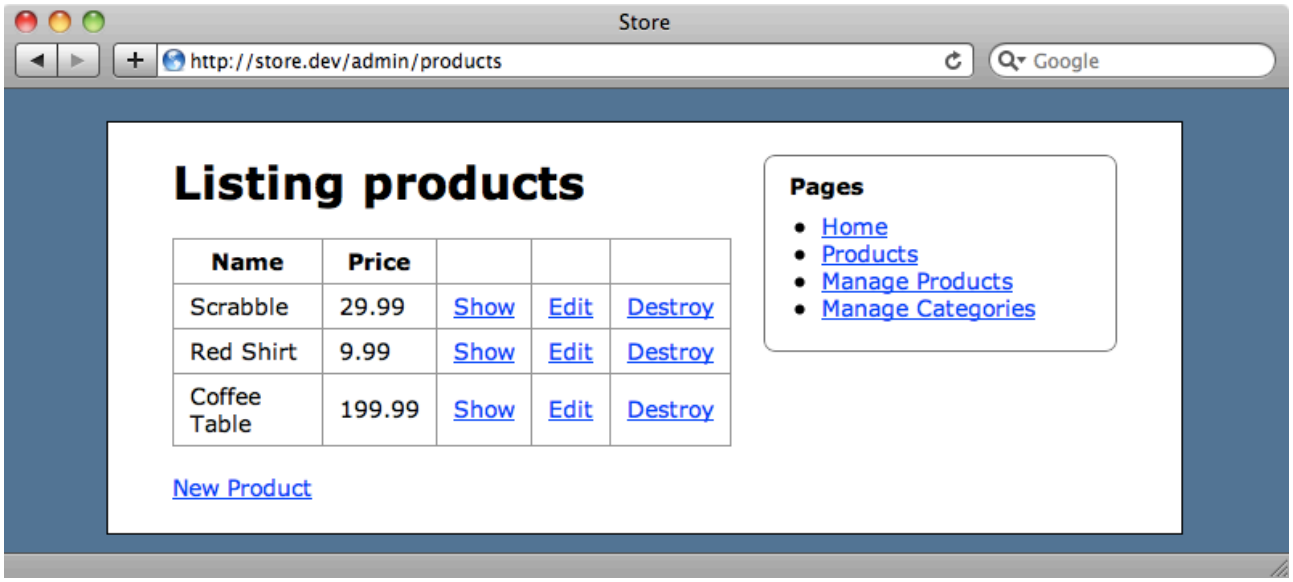
navigation will return to the default. What should we do if we want to override the navigation for all of the admin pages?

Under the `/app/views/admin` directory we have a `base` directory. We can place templates in here and they will automatically be inherited by the other controllers. We'll do so now and add another link there to distinguish this template from the other `side` templates.

`/app/views/admin/base/_side.html.erb`

```
<strong>Pages</strong>
<ul>
  <li><%= link_to "Home", root_path %></li>
  <li><%= link_to "Products", products_path %></li>
  <li><%= link_to "Manage Products", admin_products_path %></li>
  <li><%= link_to "Manage Categories", admin_categories_path %></
li>
</ul>
```

If we reload the `admin/products` page now we'll see the template from the `base` directory template applied.



This template will now be shown for any page in the `Admin` namespace as all of these pages inherit from the `Admin` namespace's `BaseController`.

Template inheritance doesn't just work for partials but for any view template. In the `app/views/admin` folder there is an `edit.html.erb` template in both the `categories` and `products` directories. Both of these contain the same code.

/app/views/admin/categories/edit.html.erb

```
<h1>Edit</h1>

<%= render 'form' %>
```

If we move one of these templates up into the `base` directory and remove the other both controllers will inherit the template from there. Both pages will still work as they did before. This is a really useful feature to have if we are creating a lot of administration pages for an application. These pages often have a lot of repetition in the views and by using this technique we can abstract these templates out into a base controller and override them when necessary in specific controllers.

While we're on the topic of overriding templates we'll show you another trick that lets you customize templates based on parameters other than the controller. Let's say that we want to provide a mobile version of the site under a `mobile` subdomain. We can change the views that are used depending on the subdomain by using view lookup paths. This feature isn't new to Rails 3.1 and works in Rails 2 as well but as it fits in well with the rest of this topic we've included it here.

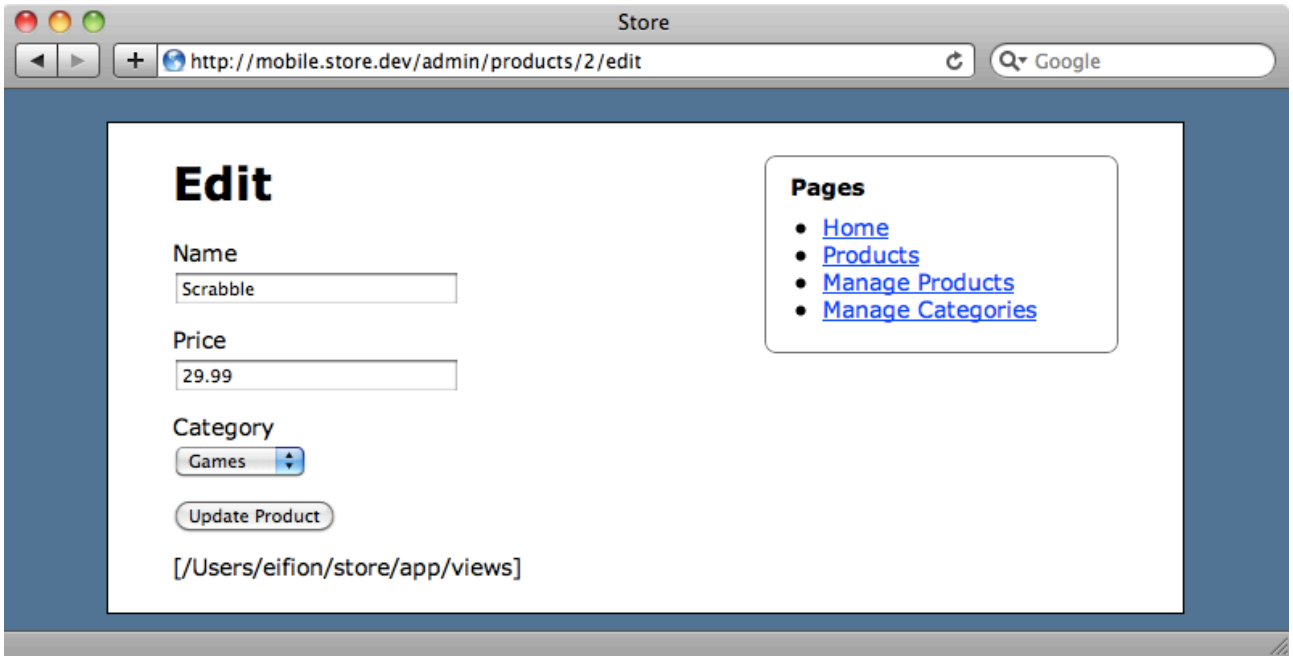
We have something called view paths in our application's controllers. If we call `controller.view_paths` in a view we can see what that controller's view paths are.

/app/views/admin/base/edit.html.erb

```
<h1>Edit</h1>

<%= render 'form' %>

<%= controller.view_paths %>
```

If we reload the edit page for a product now we'll see the `view_paths` listed. (To

make it easy to use subdomains while our application's in development we're using the Pow web server[1].)



This controller has one view path by default but we can add more. A good way to add one is by using a `before_filter` in our `ApplicationController`.

/app/controllers/application_controller.rb

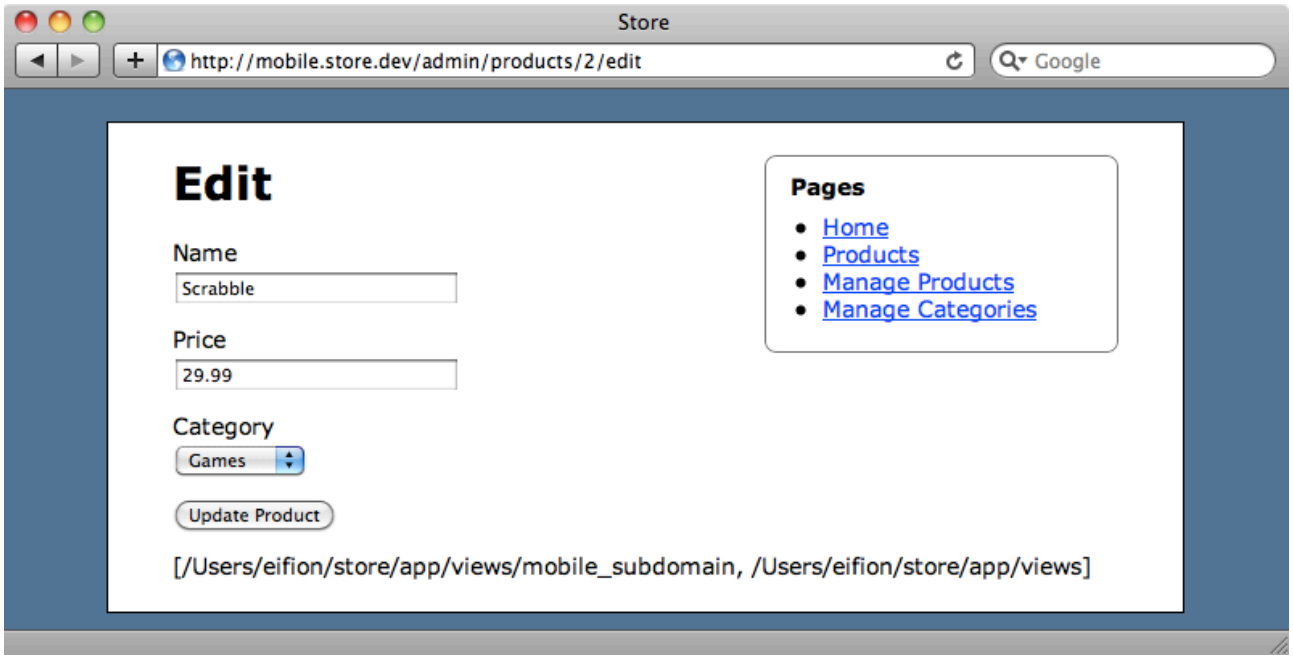
```
class ApplicationController < ActionController::Base
  protect_from_forgery
  before_filter :subdomain_view_path

  def subdomain_view_path
    if request.subdomain.present?
      prepend_view_path "app/views/#{request.subdomain}_subdomain"
    end
  end
end
```

In the `before_filter`'s method we use `prepend_view_path`[2] to add a new view path based on the request's subdomain, if a subdomain is present. We can see this new view path by reloading the page again.

---

[1] http://pow.cx/

[2] http://apidock.com/rails/AbstractController/ViewPaths/ClassMethods/prepend_view_path

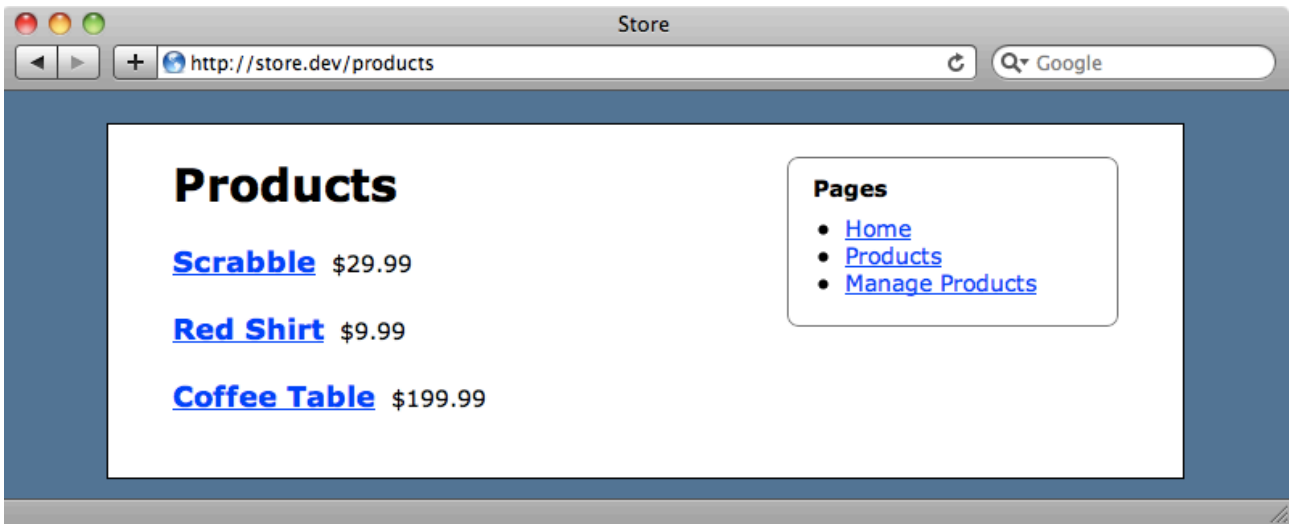


We now have the mobile subdomain listed in the `view_paths`. If we remove the `mobile` subdomain from the URL above and reload the page it will disappear. We can use this to customize views depending on the current subdomain. If we create a new directory under `views` called `mobile_subdomain` we can add view templates there that will override the equivalent default templates. We'll create a new `index.html.erb` template under a new `products` directory here and this template will only be used on this page when it's viewed under the `mobile` subdomain.

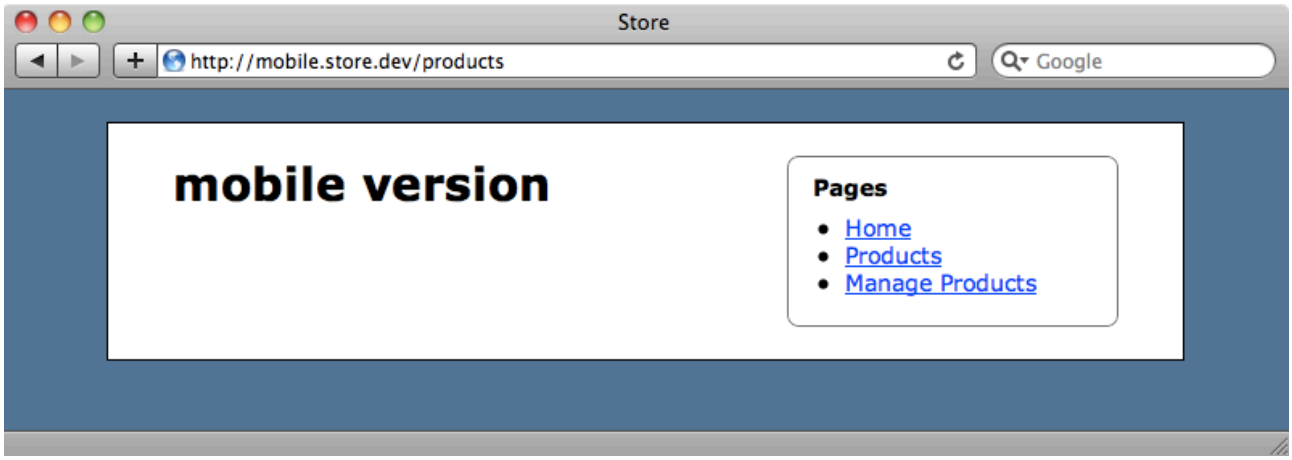
`/app/views/mobile_subdomain/products/index.html`

```
<h1>mobile version</h1>
```

If we view this page without the subdomain we'll see the default version.



When we add the `mobile` subdomain though our new template is used as the default one is overridden.



That's it for this episode on overriding templates, whether it's done through inheritance or through the view lookup path. Both of these offer a lot of customization and ways to abstract out views and override them. If you need even more customization of views you can make a view resolver but that's something for a future episode.