Chapter 15: Beautifying Pages

By Tim Condon

在上一章中，您开始使用Leaf构建一个功能强大的动态网站。然而，网页只使用简单的HTML并且没有样式 - 它们看起来不太好！在本章中，您将学习如何使用Bootstrap框架为您的页面添加样式。您还将学习如何嵌入模板，这样您只需在一个地方进行更改即可。最后，您还将了解如何使用Vapor服务文件。

嵌入模板

目前，如果更改index模板页面以添加样式，则只会影响该页面。您必须还要在缩略词详细信息页面和任何未来其他页面中复制样式。

Leaf允许您将模板嵌入到其他模板中。这使您可以创建一个“base”模板，其中包含所有页面共有的代码，并在整个站点中使用该代码。

在**Resources/Views**中创建一个新文件**base.leaf**。将**index.leaf**的内容复制到**base.leaf**中。删除<body>和</body>标签之间的所有内容。剩下的代码类似于以下内容：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>#(title) | Acronyms</title>
</head>
<body>

</body>
</html>
```

这构成了您的base模板，并且对于所有页面都是相同的。在<body>和</body>标签之间添加：

```
#get(content)
```

这使用Leaf的#get( )标签来检索content变量。要创建此变量，请打开**index.leaf**，使用以下内容替换其内容：

```
#set("content") {
  <h1>Acronyms</h1>

  #if(acronyms) {
    <table>
      <thead>
        <tr>
          <th>
            Short
          </th>
          <th>
            Long
          </th>
        </tr>
      </thead>
      <tbody>
        #for(acronym in acronyms) {
          <tr>
            <td>
              <a href="/acronyms/#(acronym.id)">
                #(acronym.short)
              </a>
            </td>
            <td>#(acronym.long)</td>
          </tr>
        }
      </tbody>
    </table>
  } else {
    <h2>There aren't any acronyms yet!</h2>
  }
}
```

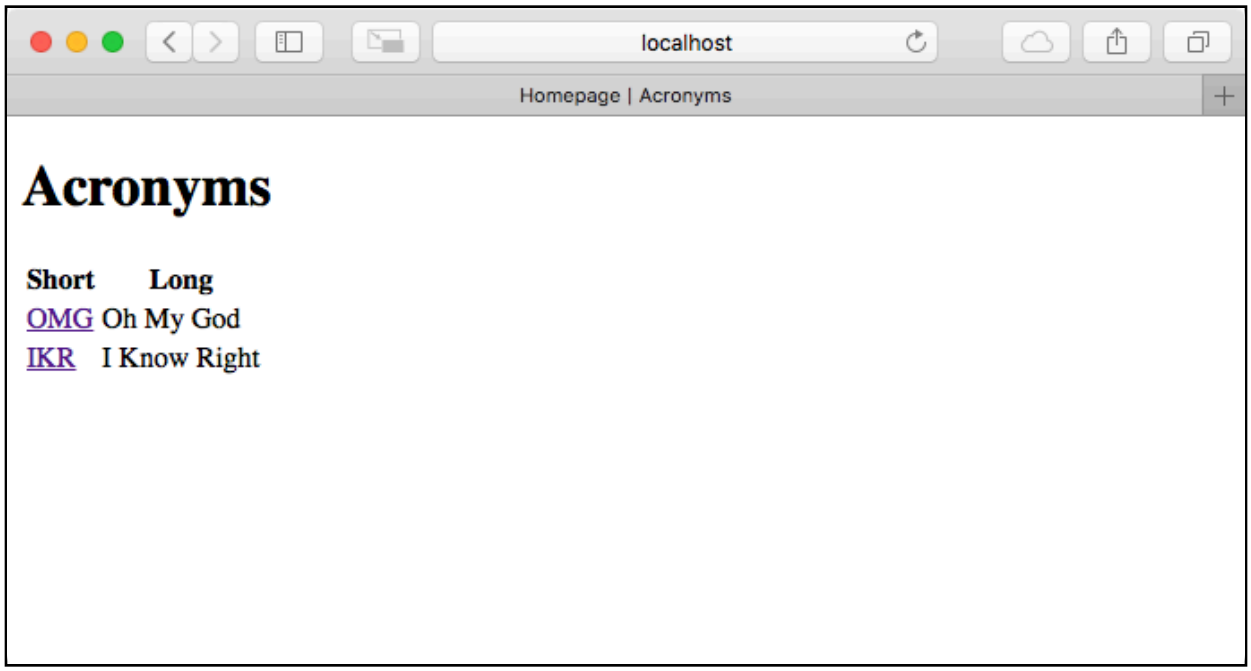
所做的更改是：

- 删除和**base.leaf**中一样的HTML。
- 使用Leaf的#set( )标签包装剩余的HTML，然后调用创建的content变量。您必须在#set( )中用引号包含变量名称，以便Leaf注册它。

最后在**index.leaf**的底部添加：

```
#embed("base")
```

这会将**base.leaf**模板嵌入到页面中并渲染它。 **base.leaf**模板使用#get( )来获取上面设置的内容。保存文件，然后构建并运行。打开浏览器并输入URL **http://localhost:8080/**。该页面渲染如以前一样：



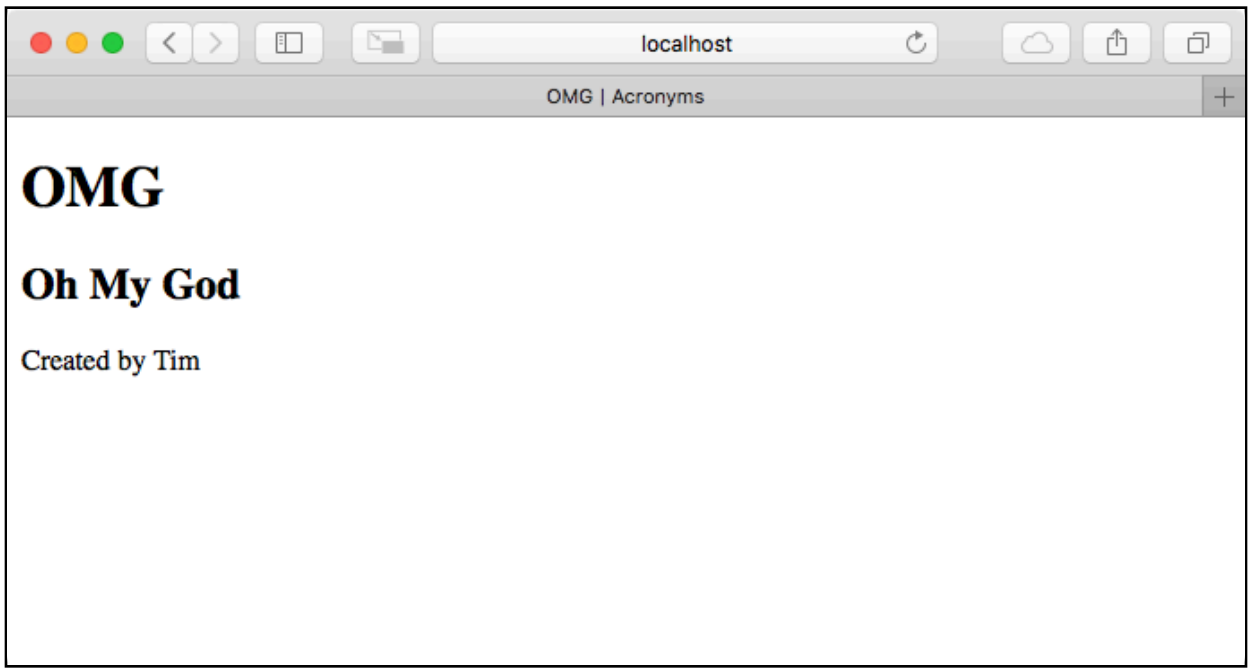
接下来，打开**acronym.leaf**并将其更改为使用base模板，方法是将其内容替换为以下内容：

```
#set("content") {
  <h1>#(acronym.short)</h1>
  <h2>#(acronym.long)</h2>

  <p>Created by #(user.name)</p>
}

#embed("base")
```

同样，所做的更改是：

- 删除所有和base模板中一样的HTML。
- 使用Leaf的#set( )标签将剩余的HTML存储在content变量中。
- 嵌入base模板以引入公共代码并渲染content。

保存文件，然后在浏览器中导航到缩略词页面。该页面使用新的base模板渲染，显示的和以前一样：



注意：在debug模式下，您可以刷新页面以获取Leaf更改。在release模式下，Leaf会缓存页面以提高性能，因此您必须重新启动应用程序才能查看更改。

Bootstrap

Bootstrap是一个开源的前端网站框架，最初由Twitter构建。它提供了网页中易于使用的各种组件。它是一个移动优先的库，可以很容易地构建一个适用于各种屏幕尺寸的网站。

要使用Bootstrap，请转到 getbootstrap.com 并单击“**Get Started**”。 Bootstrap提供了一个用于样式的CSS文件，和一些为Bootstrap组件提供功能的Javascript文件。 您需要在所有页面中包含这些文件。既然你已经创建了一个**base.leaf**模板，这就很容易做到了！

在“**Get Started**”页面上，找到“**Starter template**”部分。

在starter template的<head>部分中，复制标记为“Required meta tags”的两个<meta>标签，以及标记为“Bootstrap CSS”的CSS的<link>标签。将**base.leaf**中当前的<meta>标签替换为新标签。

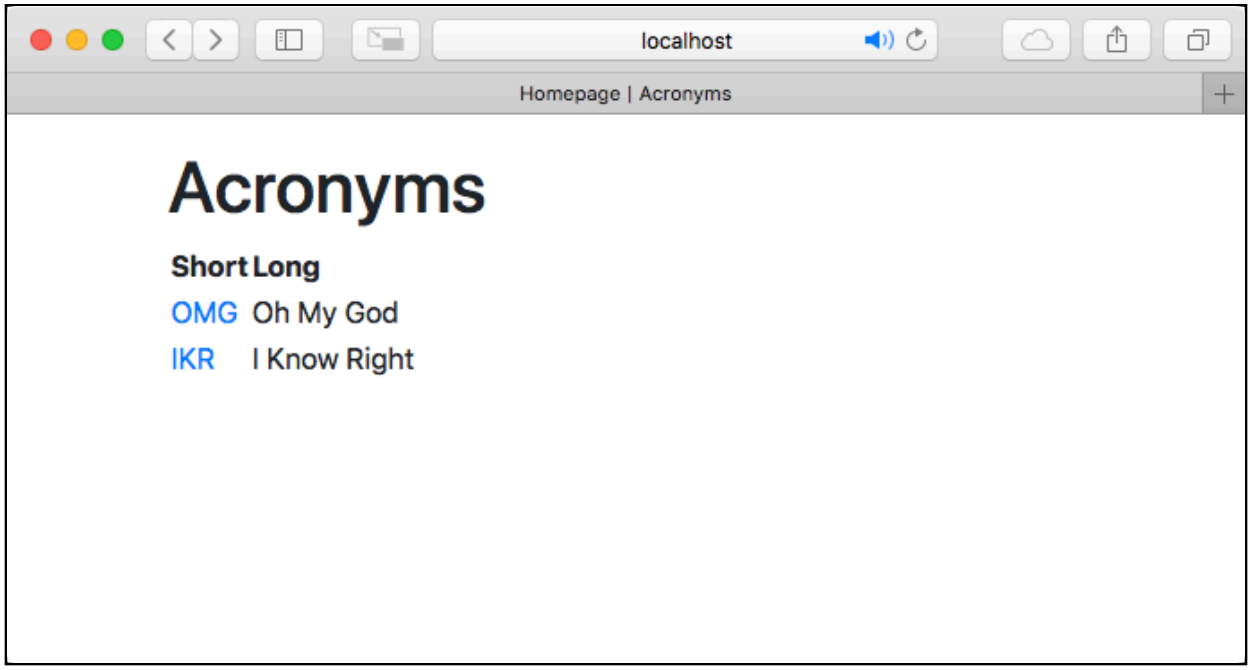
在starter template的底部，复制三个<script>标签。将它们放到**base.leaf**模板中，位于#get(content)下面和</body>标签之前。

然后保存文件，在浏览器中访问**http://:localhost:8080**。您会注意到页面看起来有点不同了。该页面现在使用了Bootstrap的样式，但您还需要添加Bootstrap特定的组件，以使您的页面真正闪耀起来。

打开**base.leaf**并用以下内容替换#get(content)：

```
<div class="container mt-3">
  #get(content)
</div>
```

这将页面的内容包装在**container**容器中，这是Bootstrap中的基本布局元素。 也在容器顶部应用了一个边距。如果您保存文件并刷新网页，您会看到页面两侧和顶部现在有一些空间，并且不再显得狭窄了：



导航

TIL网站目前包含两个页面：主页和缩略词详细页面。随着越来越多的页面被添加，在网站上找到自己的途径将变得很困难。目前，如果你转到一个缩略词的详细页面，没有简单的方法回到主页！因此添加导航到网站以使网站对用户更加友好。

HTML定义了一个<nav>元素来表示页面的导航部分。 Bootstrap提供类和实用程序来扩展它，以实现样式和移动支持。打开**base.leaf**并在 上面添加以下内容：

```
#// 1
<nav class="navbar navbar-expand-md navbar-dark bg-dark">
  #// 2
  <a class="navbar-brand" href="/">TIL</a>
  #// 3
  <button class="navbar-toggler" type="button"
   data-toggle="collapse" data-target="#navbarSupportedContent"
   aria-controls="navbarSupportedContent" aria-expanded="false"
   aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  #// 4
  <div class="collapse navbar-collapse"
   id="navbarSupportedContent">
    #// 5
    <ul class="navbar-nav mr-auto">
      #// 6
      <li class="nav-item #if(title == "Home page"){active}">
        <a href="/" class="nav-link">Home</a>
      </li>
    </ul>
  </div>
</nav>
```

这是新代码的作用：

1. 使用一些类名定义<nav>元素以进行样式设置。 Bootstrap使用这些类来指定Bootstrap导航栏，允许导航栏在中型屏幕中为全尺寸，并将黑暗主题应用于该导航栏。

2. 指定主页的根链接。

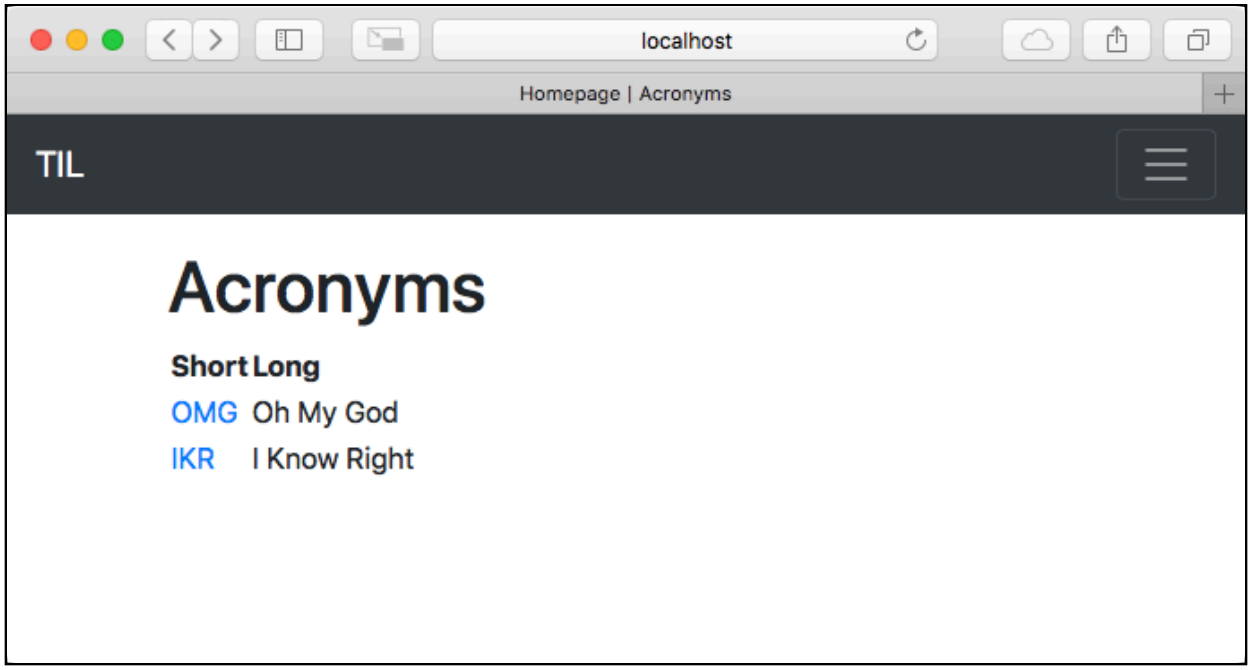
3. 创建一个按钮，切换小屏幕尺寸的导航栏。这显示并隐藏在下一个元素中定义的navbarSupportedContent部分。

4. 为小屏幕创建可折叠部分。

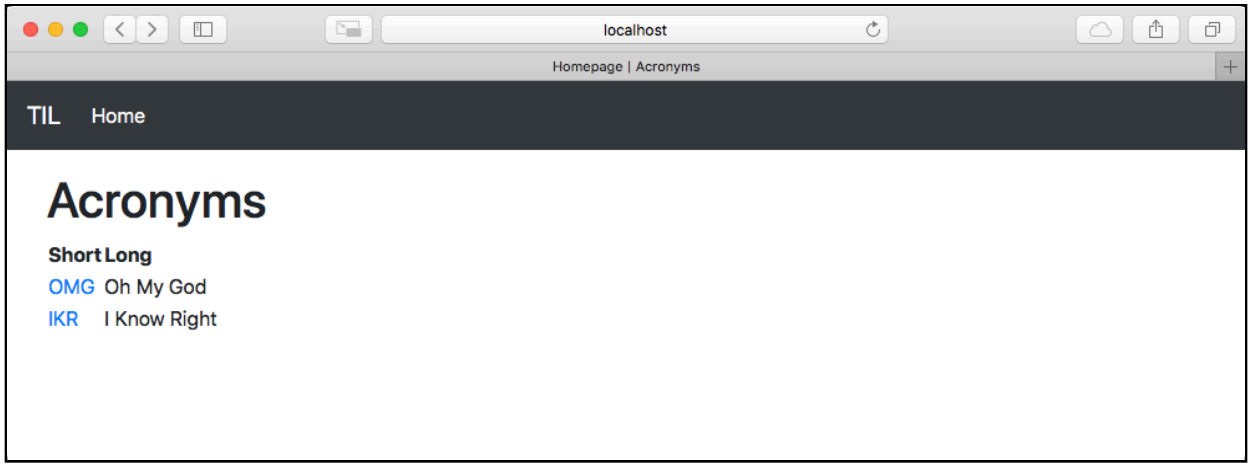
5. 定义要显示的导航链接列表。Bootstrap为导航栏设置nav-item列表项样式。而不是标准的项目列表样式。

6. 添加主页的链接。使用Leaf的#if标签来检查页面标题。如果标题设置为“Home page”，则Leaf会将active类添加到项目中。在该页面上，这会对链接设置不同的样式。

保存文件并在浏览器中刷新页面。页面开始看起来有点专业了！对于小屏幕，您将获得一个打开导航链接的切换按钮：



在较大的屏幕上，导航栏显示全部链接：



现在，当您在缩略词的详细页面时，您可以使用导航栏返回主页面！

列表

Bootstrap提供了轻松设置列表样式的类。打开**index.leaf**并用以下内容替换<table>标签：

```
<table class="table table-bordered table-hover">
```

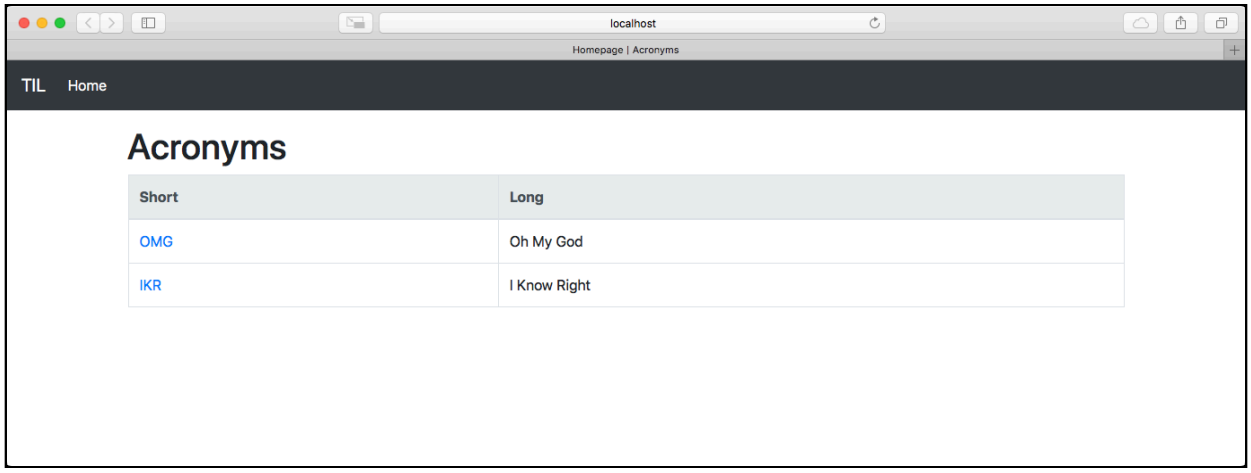
这会将以下Bootstrap类添加到列表中：

- table：应用标准的Bootstrap列表样式。
- table-bordered：为列表和单元格添加边框。
- table-hover：在表行上启用悬停样式，以便用户可以更轻松地查看他们正在查看的行。

接着，使用以下内容替换<thead>标签：

```
<thead class="thead-light">
```

这使得列表头格外突出。保存文件并刷新页面。主页现在看起来更专业了！



文件服务

几乎每个网站都需要能够托管静态文件，例如图片或样式表。大多数情况下，您将使用CDN(Content Delivery Network)或服务器(如Nginx或Apache)执行此操作。但是，Vapor提供了FileMiddleware来为文件服务。

要启用此功能，请在Xcode中打开**configure.swift**。找到// Register middleware部分，并在var middlewares = MiddlewareConfig()下面添加以下内容（如果该行已存在，则取消注释）：

```
middlewares.use(FileMiddleware.self)
```

这将FileMiddleware添加到MiddlewareConfig以提供文件服务。默认情况下，它为工程项目的**Public**目录中的文件提供服务。例如，如果在**Public/styles**目录下，你有一个名为**stylesheet.css**的文件，则可以从路径**/styles/stylesheet.css**访问该文件。

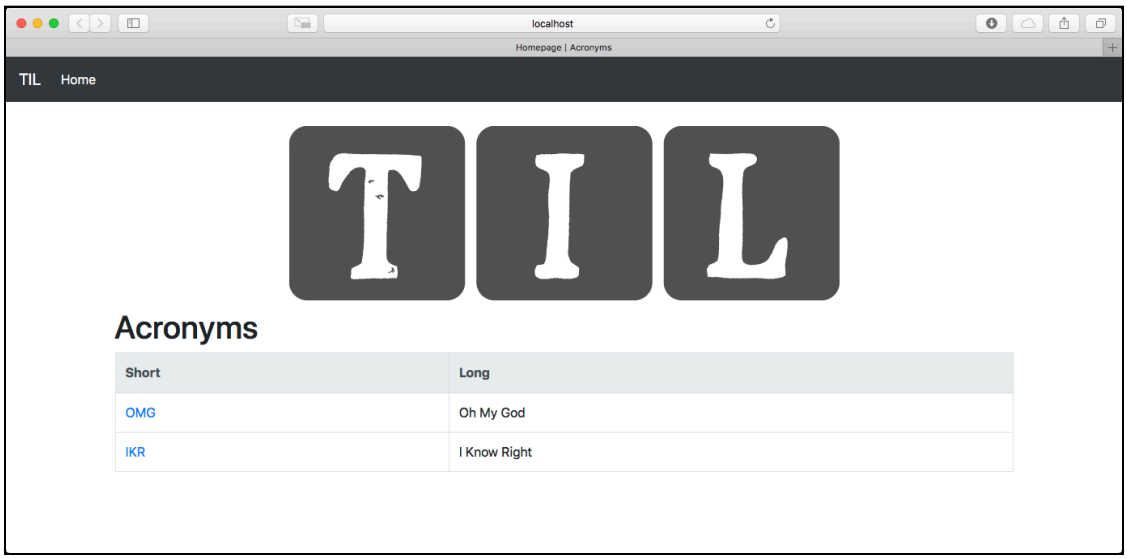
本章的入门项目包含**Public**文件夹中的**images**目录，其中包含网站的logo图标。如果您继续使用前面章节中的项目，请将**images**文件夹复制到现有的**Public**文件夹中。构建并运行，然后打开**index.leaf**。

在<h1>Acronyms</h1>上面添加以下内容：

```
<img src="/images/logo.png"
 class="mx-auto d-block" alt="TIL Logo" />
```

这会为页面添加用于图像的<img>标签。该页面加载来自**/images/logo.png**的图像，该图像对应于FileMiddleware提供的**Public/images/logo.png**。 mx-auto和d-block类告诉Bootstrap在页面中居中对齐图像。最后，alt值为图像提供了可替换的标题。帮助用户在不能加载图像时更好的理解内容。

保存文件并在浏览器中访问**http://localhost:8080**。主页现在显示图像了，对页面进行最后的修饰：



用户

该网站现在有一个显示所有缩略词的页面和一个显示缩略词详情的页面。接下来，您将添加查看所有用户和特定用户信息的页面。

在**Resources/Views**中创建一个名为**user.leaf**的新文件。像下面这样实现模板：

```
#// 1
#set("content") {
  #// 2
  <h1>#(user.name)</h1>
  #// 3
  <h2>#(user.username)</h2>

  #// 4
  #if(count(acronyms) > 0) {
    <table class="table table-bordered table-hover">
      <thead class="thead-light">
        <tr>
          <th>
            Short
          </th>
          <th>
            Long
          </th>
        </tr>
      </thead>
      <tbody>
        #// 5
        #for(acronym in acronyms) {
          <tr>
            <td>
              <a href="/acronyms/#(acronym.id)">
                #(acronym.short)</a>
            </td>
            <td>#(acronym.long)</td>
          </tr>
        }
      </tbody>
    </table>
  } else {
    <h2>There aren't any acronyms yet!</h2>
  }
}

#// 6
#embed("base")
```

这是新页面的作用：

1. 设置base模板的content变量。
2. 在<h1>标题中显示用户名。

3. 在<h2>标题中显示用户可选名称。
4. 使用Leaf的#if标签和count标签的组合来查看用户是否有任何缩略词。
5. 从注入的acronyms属性中显示缩略词表。此表与**index.leaf**模板中的表相同。
6. 嵌入base模板以引入所有公共的HTML。

在Xcode中，打开**WebsiteController.swift**。在文件的底部为用户页面创建一个新的context类型：

```
struct UserContext: Encodable {
  let title: String
  let user: User
  let acronyms: [Acronym]
}
```

此context具有以下属性：

- 页面标题，即用户名。
- 页面引用的用户对象。
- 此用户创建的缩略词。

接下来，在此页面的acronymHandler(_:)下面添加以下处理程序：

```
// 1
func userHandler(_ req: Request) throws -> Future<View> {
  // 2
  return try req.parameters.next(User.self)
    .flatMap(to: View.self) { user in
      // 3
      return try user.acronyms
        .query(on: req)
        .all()
        .flatMap(to: View.self) { acronyms in
          // 4
          let context = UserContext(
            title: user.name,
            user: user,
            acronyms: acronyms)
          return try req.view().render("user", context)
      }
  }
}
```

这是路由处理程序的作用：

1. 为返回Future <View>的用户页面定义路由处理程序。
2. 从请求的parameters中获取用户并解包future。
3. 使用计算属性获取用户的缩略词并解包future。
4. 创建一个UserContext，然后渲染**user.leaf**，返回结果。在本例中，如果缩略词数组为空，则不会将其设置为nil。这不是必需的，因为您在模板中已检查了缩略词数组的计数。

最后，在boot(router:)的末尾添加以下内容来注册此路由：

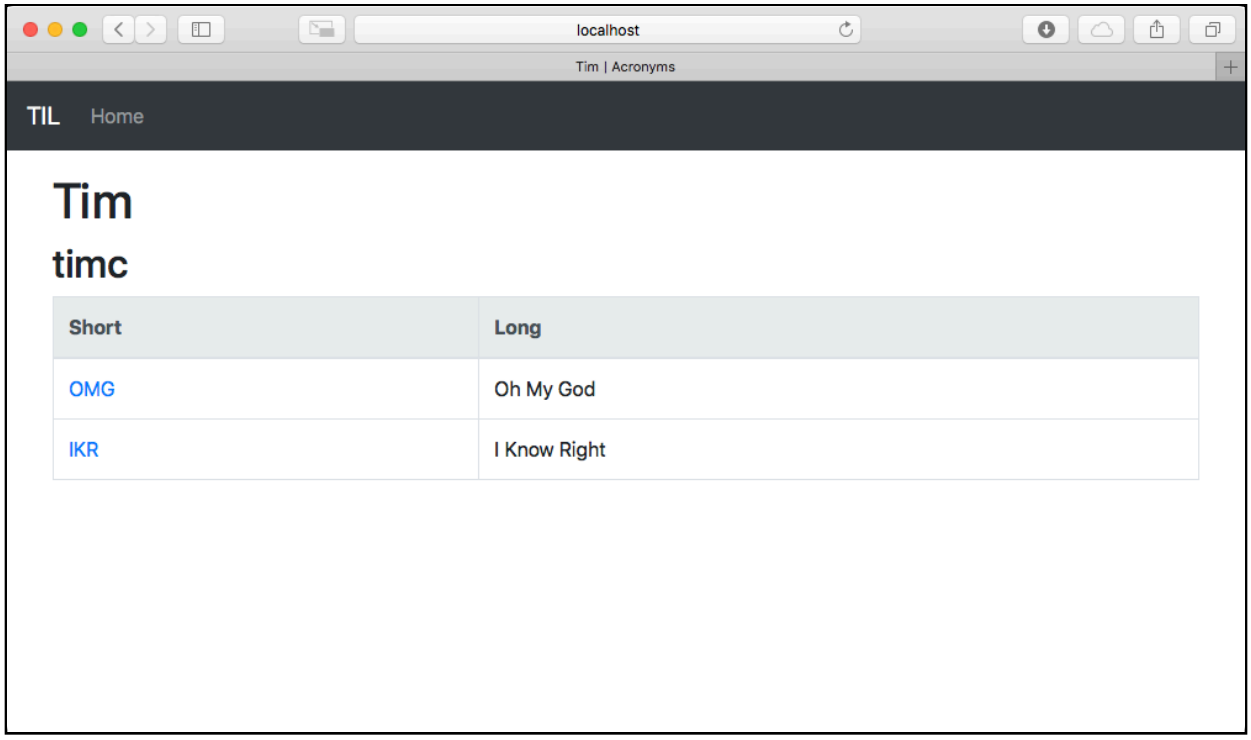
```
  router.get("users", User.parameter, use: userHandler)
```

这会为API **/users/<USER ID>**注册路由。构建并运行。

接下来，打开**acronym.leaf**，通过用以下代码替换 Created by #(user.name) ，来添加到新用户页面的链接：

```
  <p>Created by <a href="/users/#(user.id)/">#(user.name)</a></p>
```

保存文件，然后打开浏览器。转到**http://localhost:8080**并单击其中一个缩略词。该页面现在显示用户页面的链接。单击链接访问新创建的页面：



您在本章中要实现的最后一个页面是显示所有用户的列表。在**Resources/Views**中创建一个名为**allUsers.leaf**的新文件。打开文件并添加以下内容：

```
#// 1
#set("content") {

  #// 2
  <h1>All Users</h1>

  #// 3
  #if(count(users) > 0) {
    <table class="table table-bordered table-hover">
      <thead class="thead-light">
        <tr>
          <th>
            Username
          </th>
          <th>
            Name
          </th>
        </tr>
      </thead>
      <tbody>
        #for(user in users) {
          <tr>
            <td>
              <a href="/users/#(user.id)">
                #(user.username)
              </a>
            </td>
            <td>#(user.name)</td>
          </tr>
        }
      </tbody>
    </table>
  } else {
    <h2>There aren't any users yet!</h2>
  }
}

#embed("base")
```

这是新页面的作用：

1. 设置base模板的content变量。

2. 显示<h1>的标题为“All Users”。

3. 查看上下文是否提供了任何的用户。如果提供了，请创建一个包含两列的表：username和name。这类似于缩略词表。

保存文件并在Xcode中打开**WebsiteController.swift**。在文件的底部，为页面创建一个新的context：

```
struct AllUsersContext: Encodable {
  let title: String
  let users: [User]
}
```

此上下文类型包含标题和用户数组。接下来，在userHandler(_:)下面添加以下内容，为新页面创建路由处理程序：

```
// 1
func allUsersHandler(_ req: Request) throws -> Future<View> {
  // 2
  return User.query(on: req)
    .all()
    .flatMap(to: View.self) { users in
      // 3
      let context = AllUsersContext(
        title: "All Users",
        users: users)
      return try req.view().render("allUsers", context)
  }
}
```

这是新路由处理程序的作用：

1. 为“All Users”页面定义一个返回Future <View>的路由处理程序。
2. 从数据库中获取用户并解包future。
3. 创建一个AllUsersContext实例并渲染**allUsers.leaf**模板，然后返回结果。

接着，在boot(router:)底部注册路由：

```
router.get("users", use: allUsersHandler)
```

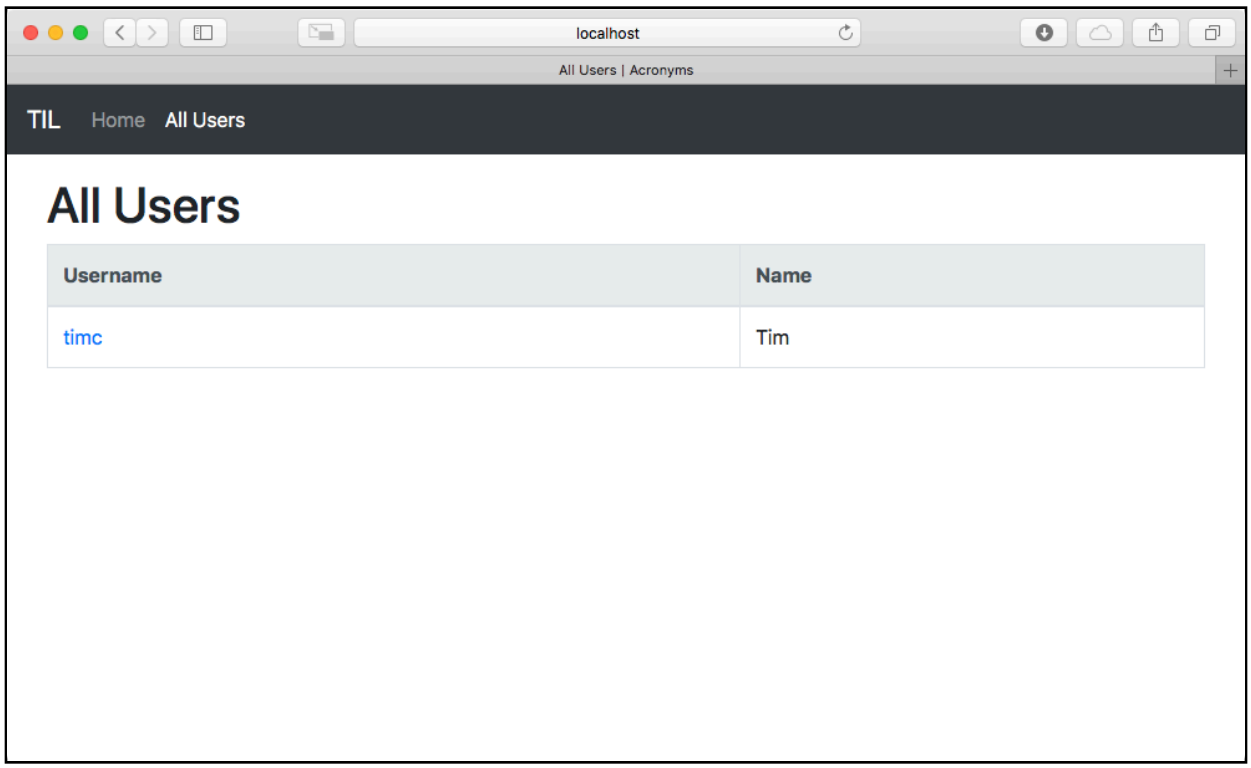
这会注册API **/users/**的路由。构建并运行，然后打开**base.leaf**。在导航栏中的
标签上方添加指向新页面的链接：

```
<li class="nav-item #if(title == "All Users"){active}">
  <a href="/users" class="nav-link">All Users</a>
</li>
```

如果页面标题为“All Users”，则会添加指向/**users**的链接并将链接设置为active。

保存文件并打开浏览器。

转到**http://localhost:8080**，您将在导航栏中看到一个新链接。单击**All Users**，您将看到新的“All Users”页面：



共享模板

本章最后要做的是重构我们的缩略词表。目前index页面和用户信息页面都使用缩写词表。但是，您已经复制了表的代码。如果要更改缩略词表，则必须在两个位置进行更改。这是模板应该要解决的问题！

在**Resources/Views**中创建一个名为**acronymsTable.leaf**的新文件。打开**user.leaf**并将列表代码复制到**acronymsTable.leaf**中。新文件应包含以下内容：

```
#if(count(acronyms) > 0) {
  <table class="table table-bordered table-hover">
    <thead class="thead-light">
      <tr>
        <th>Short</th>
        <th>Long</th>
      </tr>
    </thead>
    <tbody>
```

```
      #for(acronym in acronyms) {
        <tr>
          <td>
            <a href="/acronyms/#(acronym.id)">
              #(acronym.short)
            </a>
          </td>
          <td>#(acronym.long)</td>
        </tr>
      }
    </tbody>
  </table>
} else {
  <h2>There aren't any acronyms yet!</h2>
}
```

在**user.leaf**中，删除现已复制到**acronymsTable.leaf**中的代码，并在其中插入以下内容：

```
#embed("acronymsTable")
```

就像使用**base.leaf**一样，这会将**acronymsTable.leaf**的内容嵌入到模板中。保存文件并在浏览器中导航到用户页面 - 它应该像以前一样显示用户的缩略词。

打开**index.leaf**并删除#if(acronyms)及其中的所有代码。再次，在其位置插入以下内容：

```
#embed("acronymsTable")
```

保存文件。最后，打开**WebsiteController.swift**并更改IndexContext，这样缩略 词不再是可选的：

```
let acronyms: [Acronym]
```

acronymsTable.leaf检查数组的个数以确定是否显示列表。这更容易阅读和理解。在indexHandler(_:)中，删除acronymsData并将acronyms数组传递给IndexContext：

```
let context = IndexContext(title: "Home page", acronyms: acronyms)
```

构建并运行应用程序并在浏览器中导航到**http://localhost:8080**。所有缩略词应该都还在那里。

然后去哪儿?

现在您已经完成了这一章，TIL应用程序的网站看起来好多了！使用Bootstrap框架可以让您轻松地设计站点的样式。这会给访问您的应用程序的用户留下更好的印象。

在接下来的章节中，您将学习如何从仅显示在页面上的信息到实现能够创建缩略词，类别和用户的所有功能。