Chapter 16: Making a Simple Web App, Part 1

By Tim Condon

在前面的章节中，您学习了如何在网站中显示数据以及如何使用Bootstrap使页面看起来更漂亮。在本章中，您将学习如何创建不同的模型以及如何编辑缩略词。

类别

您已创建了用于查看缩略词和用户的页面。现在是时候为类别创建类似的页面了。打开**WebsiteController.swift**。在文件的底部，为“All Categories”页面添加上下文数据结构：

```
struct AllCategoriesContext: Encodable {
  // 1
  let title = "All Categories"
  // 2
  let categories: Future<[Category]>
}
```

这是它的作用：

1. 定义模板的页面标题。

2. 定义要在页面中显示的类别Future数组。

Leaf知道如何处理futures。这有助于整理代码，因为那时您不需要访问请求处理程序中已解析的futures。

接下来，在allUsersHandler(_:)下面添加以下内容，为“All Categories”页面创建一个新的路由处理程序：

```
func allCategoriesHandler(_ req: Request) throws
  -> Future<View> {
```

```
  // 1
  let categories = Category.query(on: req).all()
  let context = AllCategoriesContext(categories: categories)
  // 2
  return try req.view().render("allCategories", context)
}
```

以下是此路由处理程序的作用：

1. 创建一个AllCategoriesContext。注意，上下文直接包含查询结果，因为Leaf可以处理futures。

2. 使用提供的上下文渲染**allCategories.leaf**模板。

在**Resources/Views**中为“All Categories”页面创建一个名为**allCategories.leaf**的新文件。打开新文件并添加以下内容：

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

  <h1>All Categories</h1>

  #// 2
  #if(count(categories) > 0) {
    <table class="table table-bordered table-hover">
      <thead class="thead-light">
        <tr>
          <th>
            Name
          </th>
        </tr>
      </thead>
      <tbody>
        #// 3
        #for(category in categories) {
          <tr>
            <td>
              <a href="/categories/#(category.id)">
                #(category.name)
              </a>
            </td>
          </tr>
        }
      </tbody>
    </table>
  } else {
    <h2>There aren't any categories yet!</h2>
  }
}

#embed("base")
```

这个模板就像所有缩略词的列表一样，但重点是：

1. 设置content变量以供**base.leaf**使用。

2. 查看是否存在任何类别。您可以使用与non-futures变量完全相同的方式访问future变量。 Leaf使这个对于模板来说是透明的。

3. 循环遍历每个类别，并使用类别名称向表中添加行，每行都链接到类别详细页面。

现在，您需要一种方法来显示类别中的所有缩略词。打开**WebsiteController.swift**并在新类别页面的文件底部添加以下上下文数据类型：

```
struct CategoryContext: Encodable {
  // 1
  let title: String
  // 2
  let category: Category
  // 3
  let acronyms: Future<[Acronym]>
}
```

以下是上下文包含的内容：

1. 页面标题；您将其设置为类别名称。

2. 页面的类别。这不是Future <Category>，因为您需要类别的名称来设置标题。这意味着您必须在路由处理程序中解包future。

3. 该类别的缩略词，作为future提供。

接下来，在allCategoriesHandler(_:)下面添加以下内容，为页面创建路由处理程序：

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

这是路由处理程序的作用：

1. 从请求的参数中获取类别并解包返回的future。
2. 创建查询以获取该类别的所有缩略词。这是一个 Future<[Acronym]>。
3. 为页面创建上下文。
4. 返回使用**category.leaf**模板渲染的视图。

在**Resources/Views**中创建新模板**category.leaf**。打开新文件并添加以下内容：

```
#set("content") {
  <h1>#(category.name)</h1>

  #embed("acronymsTable")
}

#embed("base")
```

这与用户的页面几乎相同，只是标题是类别名称。注意，您正在使用**acronymsTable.leaf**模板将表显示为缩略词。这避免了复制另一个表，这再次显示了模板的强大功能。打开**base.leaf**并在链接到所有用户页面后面添加以下内容：

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

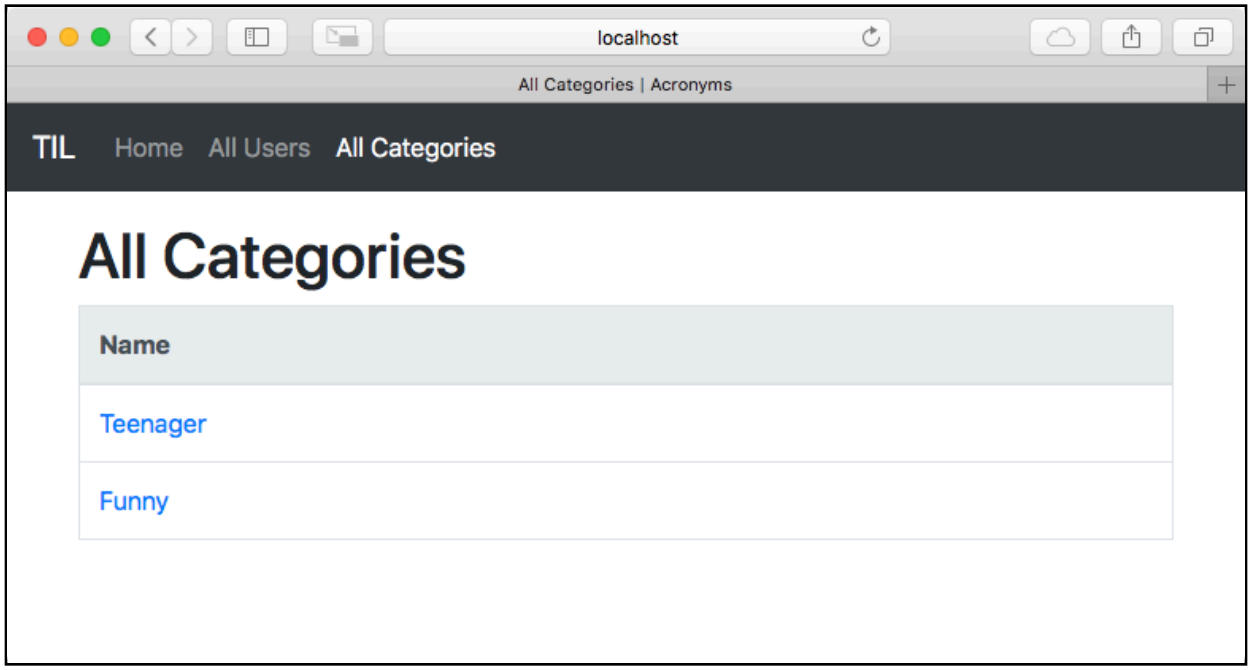
这会为所有类别页面添加指向站点导航的新链接。最后打开**WebsiteController.swift**并在boot(router:)的末尾，添加以下内容以注册新路由：

```
// 1
router.get("categories", use: allCategoriesHandler)
// 2
router.get(
  "categories", Category.parameter,
  use: categoryHandler)
```

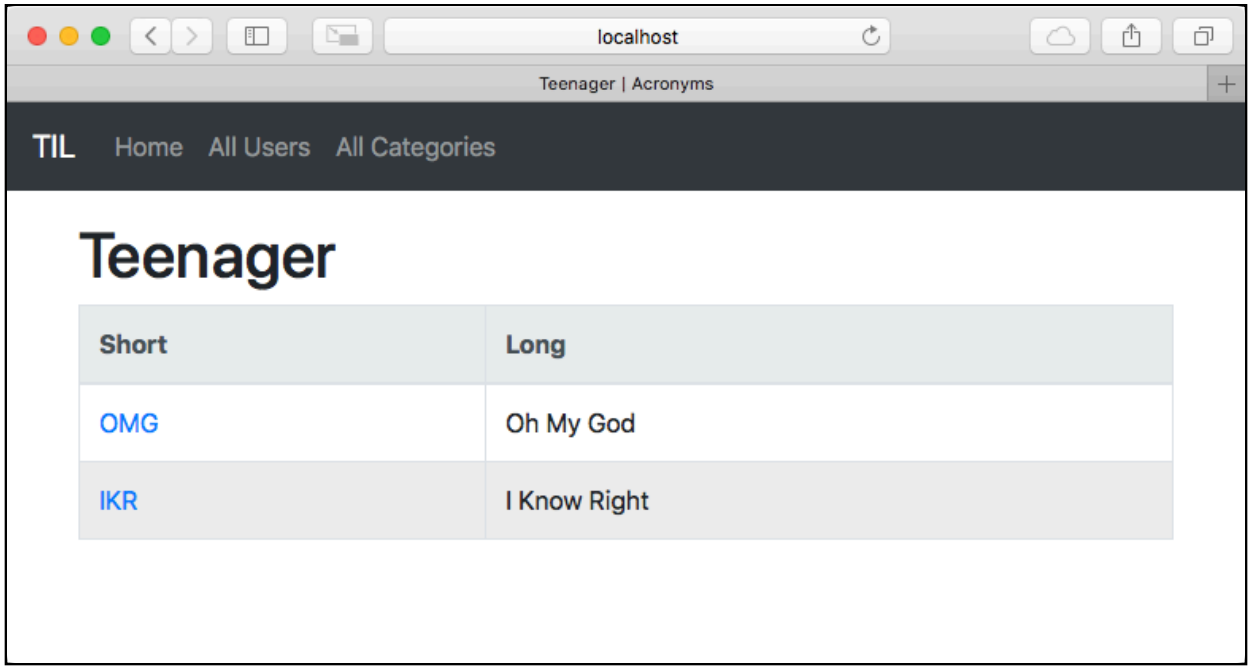
这是它的作用：

1. 在**/categories**注册接受GET请求并调用allCategoriesHandler(_:)的路由。
2. 在**/categories/<CATEGORY ID>**注册接受GET请求并调用categoryHandler(_:)的路由。

构建并运行，然后在浏览器中转到**http://localhost:8080/**。单击菜单中的新的“**All Categories**”链接，您将转到新的“All Categories”页面：



单击一个类别，您将看到包含该类别所有缩略词的类别信息页面：



创建缩略词

要在Web应用程序中创建缩略词，您必须要实现两个路由。您处理GET请求以显示要填写的表单。然后，您处理POST请求以接受表单发送的数据。创建缩略词的页面需要所有用户的列表，以允许选择哪个用户拥有该缩略词。在**WebsiteController.swift**的底部创建一个上下文来表示：

```
struct CreateAcronymContext: Encodable {
  let title = "Create An Acronym"
  let users: Future<[User]>
}
```

您再次在上下文中使用Future。接下来，创建一个路由处理程序，以在categoryHandler(_:)下面显示“Create An Acronym”页面：

```
func createAcronymHandler(_ req: Request) throws
  -> Future<View> {
  // 1
  let context = CreateAcronymContext(
    users: User.query(on: req).all())
  // 2
  return try req.view().render("createAcronym", context)
}
```

这是它的作用：

1. 通过传入查询获取所有用户来创建上下文。

2. 使用**createAcronym.leaf**模板渲染页面。

接下来，在createAcronymHandler(_:)下面添加以下内容，为POST请求创建路由处理程序：

```
// 1
func createAcronymPostHandler(
  _ req: Request,
  acronym: Acronym
) throws -> Future<Response> {
  // 2
  return acronym.save(on: req)
    .map(to: Response.self) { acronym in
      // 3
      guard let id = acronym.id else {
        throw Abort(.internalServerError)
      }
      // 4
      return req.redirect(to: "/acronyms/\(id)")
  }
}
```

这是它的作用：

1. 声明一个以Acronym作为参数的路由处理程序。 Vapor自动将表单数据解码为Acronym对象。

2. 保存提供的缩略词并解包返回的future。

3. 确保已设置ID，否则抛出**500 Internal Server Error**。

4. 重定向到新创建的缩略词的页面。

接下来，要注册这些路由，将以下内容添加到boot(router:)的底部：

```
// 1
router.get("acronyms", "create", use: createAcronymHandler)
// 2
router.post(
  Acronym.self, at: "acronyms", "create",
  use: createAcronymPostHandler)
```

这是代码的作用：

1. 在**/acronyms/create**注册接受GET请求并调用createAcronymHandler(_:)的路由。

2. 在**/acronyms/create**注册一个接受POST请求并调用createAcronymPostHandler(_:acronym:)的路由。这也将请求body解码为Acronym。

您现在需要一个模板来显示创建缩略词表单。在**Resources/Views**中创建一个名为**createAcronym.leaf**的新文件。打开文件并添加以下内容：

```
#// 1
#set("content") {
  <h1>#(title)</h1>

  #// 2
  <form method="post">
    #// 3
    <div class="form-group">
      <label for="short">Acronym</label>
      <input type="text" name="short" class="form-control"
       id="short"/>
    </div>

    #// 4
    <div class="form-group">
      <label for="long">Meaning</label>
      <input type="text" name="long" class="form-control"
       id="long"/>
    </div>

    <div class="form-group">
```

```
      <label for="userID">User</label>
      #// 5
      <select name="userID" class="form-control" id="userID">
        #// 6
        #for(user in users) {
          <option value="#(user.id)">
            #(user.name)
          </option>
        }
      </select>
    </div>

    #// 7
    <button type="submit" class="btn btn-primary">
      Submit
    </button>
  </form>
}

#embed("base")
```

这是模板的作用：

1. 定义基本模板中使用的content变量。

2. 创建一个HTML表单。将方法设置为POST。这意味着当用户提交表单时，浏览器使用POST请求将数据发送到同一URL。

3. 为缩略词的short值创建一个表单组。使用HTML的<input>元素允许用户插入文本。name属性告诉浏览器在请求中发送数据时此输入元素的键应该是什么。

4. 使用HTML的<input>元素为缩略词的long值创建一个表单组。

5. 为缩略词的用户创建一个表单组。使用HTML的<select>元素显示不同用户的下拉菜单。

6. 使用Leaf的#for()循环遍历提供的用户，并在<select>上添加每个用户选项。

7. 创建一个提交按钮，用户可以单击该按钮将表单发送到您的Web应用程序。

最后，在**base.leaf**中的
标记之前的添加指向新页面的链接：

```
#// 1
<li class="nav-item #if(title == "Create An Acronym"){active}">
  #// 2
  <a href="/acronyms/create" class="nav-link">
    Create An Acronym
  </a>
</li>
```

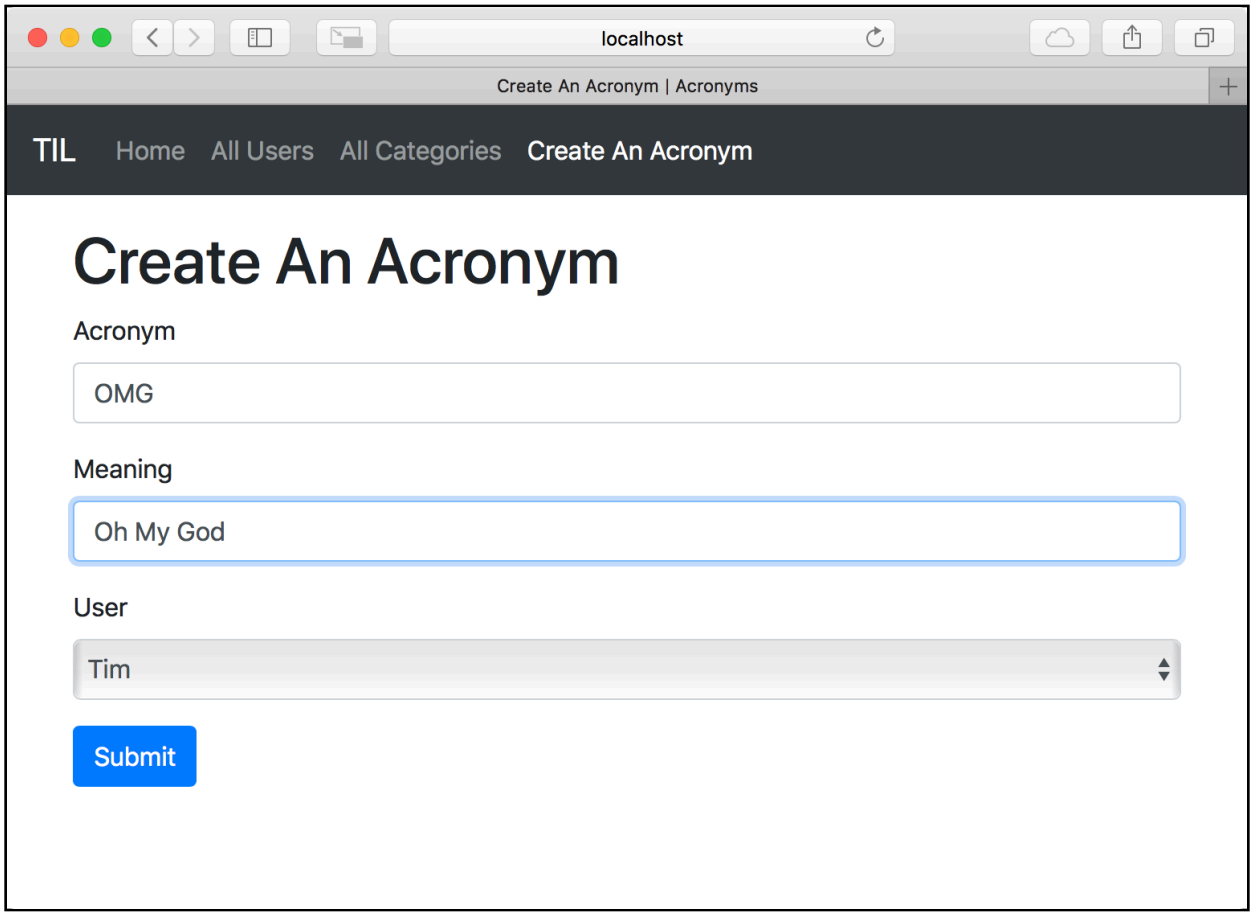
这是代码的作用：

1. 将新导航项添加到导航栏。如果您在“Create An Acronym”页面上，将该项标记为活动状态。

2. 添加指向创建页面的链接。

构建并运行，然后打开浏览器。导航到**http://localhost:8080**，您将在导航栏中看到一个新选项“Create An Acronym”。单击链接转到新页面。填写表单并单击“**Submit**”。

该应用程序将您重定向到新的缩略词的页面：



编辑缩略词

您现在知道如何通过网站创建缩略词。但是怎么样编辑缩略词呢？感谢Leaf，您可以重用许多相同的组件来允许用户编辑缩略词。打开**WebsiteController.swift**。

在文件末尾，添加以下上下文数据类型以编辑缩略词：

```
struct EditAcronymContext: Encodable {
  // 1
  let title = "Edit Acronym"
  // 2
  let acronym: Acronym
  // 3
  let users: Future<[User]>
  // 4
  let editing = true
}
```

以下是上下文包含的内容：

1. 页面标题：“Edit Acronym”。
2. 编辑的缩略词。
3. 将在表单中显示的用户Future数组。
4. 一个标志为告诉模板该页面用于编辑缩略词。

接下来，在createAcronymPostHandler(_:acronymn:)下面添加以下路由处理程序以显示编辑缩略词表单：

```
func editAcronymHandler(_ req: Request) throws -> Future<View> {
  // 1
  return try req.parameters.next(Acronym.self)
    .flatMap(to: View.self) { acronym in
      // 2
      let context = EditAcronymContext(
        acronym: acronym,
        users: User.query(on: req).all())
      // 3
      return try req.view().render("createAcronym", context)
  }
}
```

这是这个路由的作用：

1. 从请求的参数中获取要编辑的缩略词并解包future。
2. 创建一个上下文来编辑缩略词，传入所有用户。

3. 使用**createAcronym.leaf**模板渲染页面，该模板与创建页面使用的模板相同。

接下来，在editAcronymHandler(_:)下面为编辑缩略词页面的POST请求添加以下路由处理程序：

```
func editAcronymPostHandler(_ req: Request) throws
  -> Future<Response> {
  // 1
  return try flatMap(
    to: Response.self,
    req.parameters.next(Acronym.self),
    req.content.decode(Acronym.self)
  ) { acronym, data in
    // 2
    acronym.short = data.short
    acronym.long = data.long
    acronym.userID = data.userID

    // 3
    guard let id = acronym.id else {
      throw Abort(.internalServerError)
    }
    let redirect = req.redirect(to: "/acronyms/\(id)")
    // 4
    return acronym.save(on: req).transform(to: redirect)
  }
}
```

这是路由的作用：

1. 使用flatMap的便利方法从请求的参数中获取缩略词，和解码传入的数据，并解包这两个结果。

2. 使用新数据更新缩略词。

3. 确保已设置ID，否则抛出**500 Internal Server Error**。

4. 保存结果并转换结果以重定向到已更新的缩略词的页面。

接下来，在boot(router:)底部添加以下内容以注册两个新路由：

```
router.get(
  "acronyms", Acronym.parameter, "edit",
  use: editAcronymHandler)
router.post(
  "acronyms", Acronym.parameter, "edit",
  use: editAcronymPostHandler)
```

这会注册调用editAcronymHandler(_:)的路由 **/acronyms/<ACRONYM ID>/edit**，以接受的GET请求。它还注册了一个路由来处理对调用editAcronymPostHandler(_:)的同一URL的POST请求。

打开**createAcronym.leaf**并更改模板以适应编辑缩略词。首先，替换缩略词short的输入元素以适应编辑：

```
<input type="text" name="short" class="form-control"
 id="short" #if(editing){value="#(acronym.short)"}/>
```

如果设置了editing标志，则将<input>的value属性设置为缩略词的short属性。这是您预先填写表单进行编辑的方式。为缩略词的long输入做同样的事情：

```
<input type="text" name="long" class="form-control"
 id="long" #if(editing){value="#(acronym.long)"}/>
```

替换用户的<select>选项进行编辑：

```
<option value="#(user.id)"
 #if(editing){#if(acronym.userID == user.id){selected}}>
  #(user.name)
</option>
```

如果用户的ID与缩略词的userID匹配，则设置<option>的selected属性。这使得下拉菜单中的该选项显示为选定的选项。接下来，替换提交表单的按钮：

```
<button type="submit" class="btn btn-primary">
  #if(editing){Update} else{Submit}
</button>
```

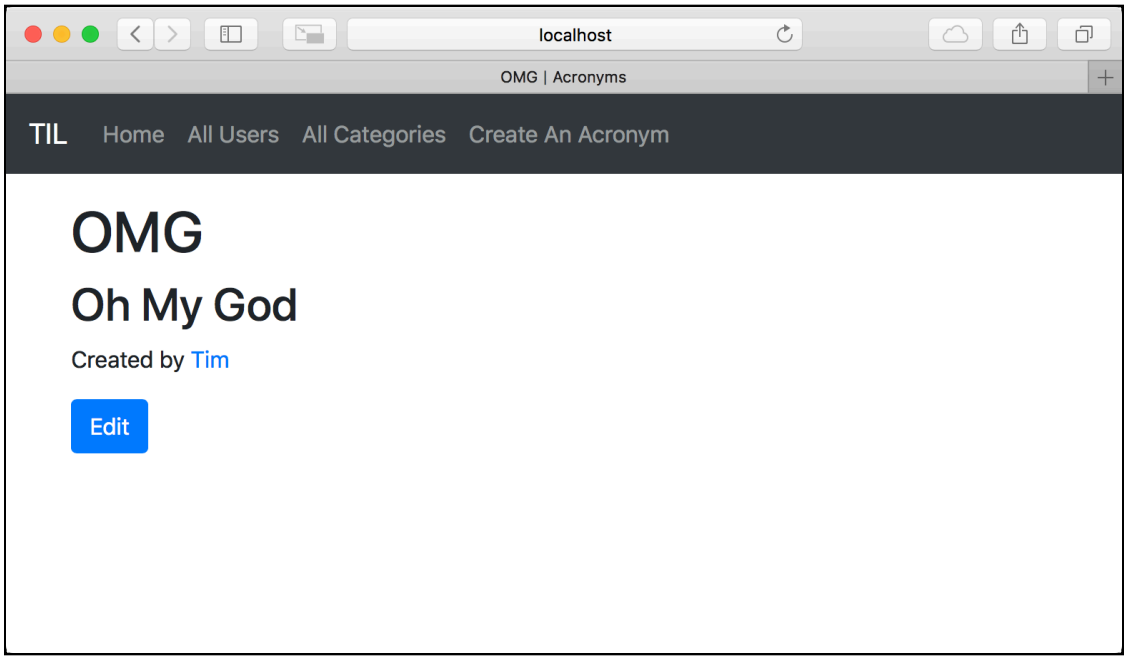
这使用Leaf的#if()/else标签将按钮的文本设置为“Update”或“Submit”，具体取决于页面的模式。

最后，打开**acronym.leaf**并在#set("content")底部添加一个按钮来编辑缩略词：

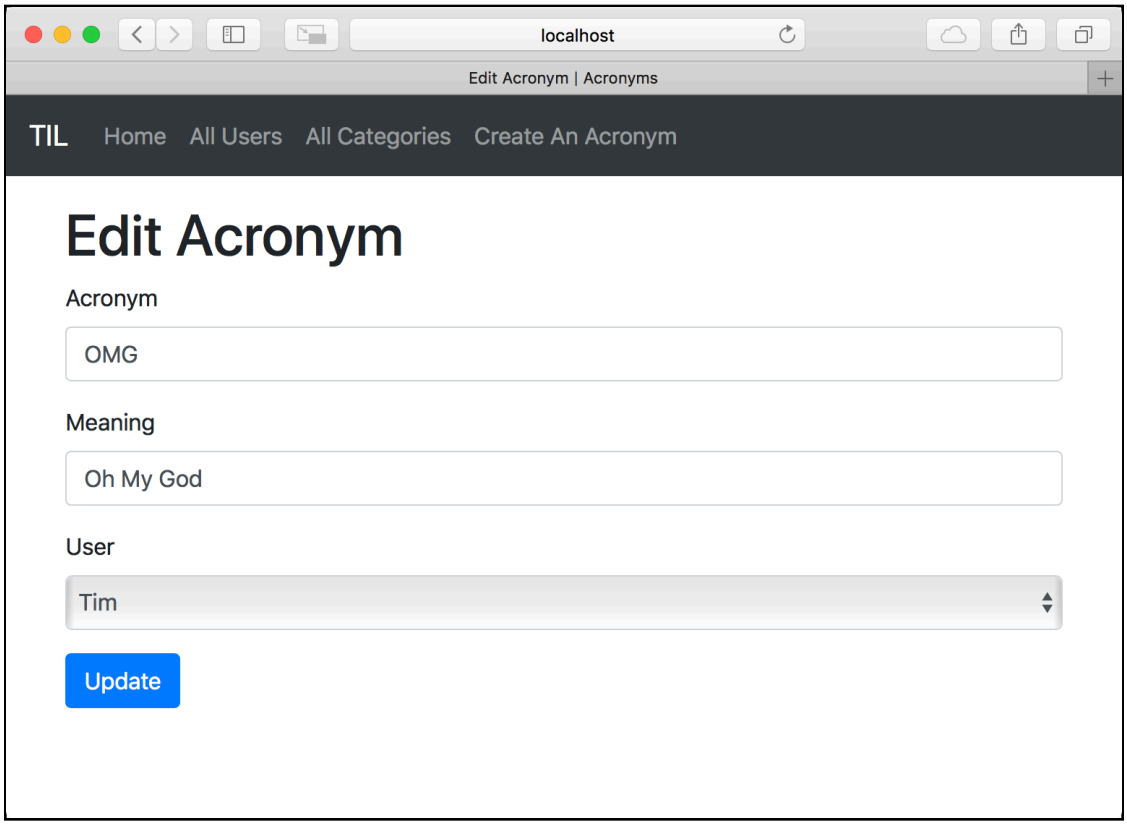
```
<a class="btn btn-primary" href="/acronyms/#(acronym.id)/edit"
 role="button">Edit</a>
```

这将创建一个指向**/acronyms/<ACRONYM ID>/edit**的HTML链接，并使用Bootstrap将链接设置为按钮样式。保存文件，在Xcode中构建并运行应用程序。在您的浏览器中打开**http://localhost:8080/**。

打开一个缩略词页面，现在底部有一个**Edit**按钮：



单击**Edit**以转到编辑缩略词页面，其中包含预先填充的所有信息。标题和按钮也不同：



更改缩略词，然后单击“**Update**”。该应用程序将您重定向到缩略词的页面，您将看到更新的信息。

删除缩略词

与创建和编辑缩略词不同，删除缩略词只需要一条路由。但是，对于Web浏览器，没有简单的方法来发送DELETE请求。浏览器只能发送请求页面的GET请求，和发送带有表单数据的POST请求。

> 可以使用JavaScript发送DELETE请求，但这超出了本章的范围。

要解决此问题，您将向删除路由发送POST请求。打开**WebsiteController.swift**并在editAcronymPostHandler(_:)下面添加以下路由处理程序以删除缩略词：

```
func deleteAcronymHandler(_ req: Request) throws
  -> Future<Response> {
  return try req.parameters.next(Acronym.self).delete(on: req)
    .transform(to: req.redirect(to: "/"))
}
```

此路由从请求的参数中提取缩略词，并在缩略词上调用delete(on:)。然后路由转换结果以将页面重定向到主界面。在boot(router:)底部注册路由：

```
router.post(
  "acronyms", Acronym.parameter, "delete",
  use: deleteAcronymHandler)
```

这会注册**/acronyms/<ACRONYM ID>/delete**路由以接受POST请求并调用deleteAcronymHandler(_:)。构建并运行。打开**acronym.leaf**并用以下内容替换编辑按钮：

```
#// 1
<form method="post" action="/acronyms/#(acronym.id)/delete">
  #// 2
  <a class="btn btn-primary" href="/acronyms/#(acronym.id)/edit"
   role="button">Edit</a> 
  #// 3
  <input class="btn btn-danger" type="submit" value="Delete" />
</form>
```

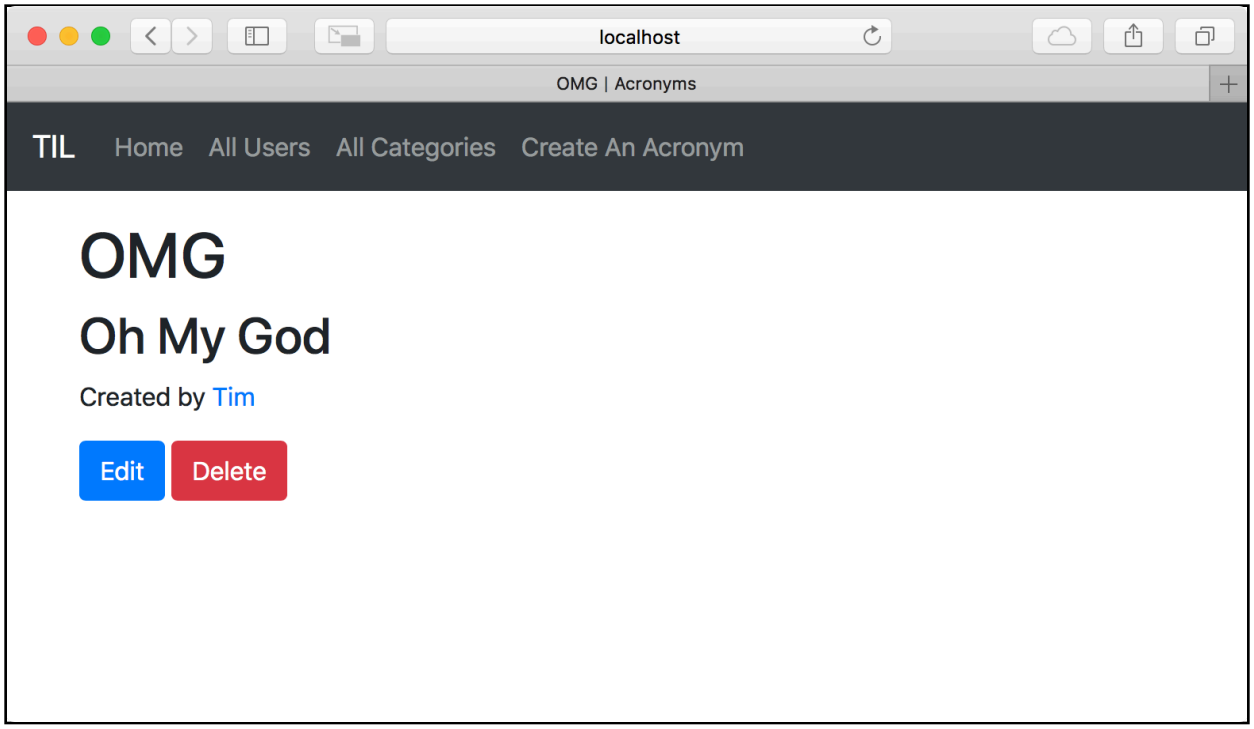
这是新代码的作用：

1. 声明发送POST请求的表单。将action属性设置为**/acronyms/<ACRONYM ID>/delete**。对于修改数据库的操作（例如创建或删除）使用POST请求是一种很好的做法。例如，这使您可以使用CSRF（跨站点请求伪造）tokens保护它们。

2. 合并页面上已存在的编辑按钮。这允许Bootstrap对齐它们。使用Bootstrap的按钮样式，使按钮看起来一样。

3. 为删除表单创建一个提交按钮。

保存文件，然后在浏览器中打开**http://localhost:8080/**。打开缩略词页面，你会看到删除按钮：



单击“**Delete**”以删除缩略词。该应用程序将您重定向到主页，并且不再显示已删除的缩略词。

然后去哪儿?

在本章中，您学习了如何显示类别以及如何创建，编辑和删除缩略词。您仍然需要完成对类别的支持，允许您的用户将缩略词分类并移除它们。您将在下一章学习如何做到这一点！