

Chapter 21: Validation

By Tim Condon

在前面的章节中，您构建了一个功能齐全的API和网站。用户可以发送请求并填写表单以创建缩略词、类别和其他用户。在本章中，您将学习如何使用Vapor的验证库来验证用户发送至应用程序的一些信息。您将在网站上创建一个注册页面供用户注册。最后，您将验证此表单中的数据，并在数据不正确时显示错误消息。

注册页面

在**Resources/Views**中创建一个名为**register.leaf**的新文件。这是注册页面的模板。打开**register.leaf**并添加以下内容：

```
#set("content") {  
  <h1>#{title}</h1>  
  
  <form method="post">  
    <div class="form-group">  
      <label for="name">Name</label>  
      <input type="text" name="name" class="form-control"  
        id="name"/>  
    </div>  
  
    <div class="form-group">  
      <label for="username">Username</label>  
      <input type="text" name="username" class="form-control"  
        id="username"/>  
    </div>  
  
    <div class="form-group">  
      <label for="password">Password</label>  
      <input type="password" name="password"  
        class="form-control" id="password"/>  
    </div>  
  </form>  
}
```

```
<div class="form-group">
  <label for="confirmPassword">Confirm Password</label>
  <input type="password" name="confirmPassword"
    class="form-control" id="confirmPassword"/>
</div>

<button type="submit" class="btn btn-primary">
  Register
</button>
</form>
}

#embed("base")
```

这与用于创建缩略词和登录的模板非常相似。该模板包含四个输入字段：

- name
- username
- password
- password confirmation

保存文件。接下来，打开**WebsiteController.swift**并在文件底部为注册页面添加以下上下文类型：

```
struct RegisterContext: Encodable {
  let title = "Register"
}
```

接下来，在**logoutHandler(_:)**下面，为注册页面添加以下路由处理程序：

```
func registerHandler(_ req: Request) throws -> Future<View> {
  let context = RegisterContext()
  return try req.view().render("register", context)
}
```

与其他路由处理程序一样，这会创建一个上下文，然后调用**render(_:_:)**来渲染**register.leaf**。

接下来，在**WebsiteController.swift**的底部，为注册创建POST请求的Content：

```
struct RegisterData: Content {
  let name: String
  let username: String
  let password: String
  let confirmPassword: String
}
```

此Content类型匹配从注册POST请求收到的预期数据。变量与**register.leaf**中的输入名称匹配。接下来，在registerHandler(:)-之后添加以下内容，为此POST请求创建路由处理程序：

```
// 1
func registerPostHandler(
    _ req: Request,
    data: RegisterData
) throws -> Future<Response> {
    // 2
    let password = try BCrypt.hash(data.password)
    // 3
    let user = User(
        name: data.name,
        username: data.username,
        password: password)
    // 4
    return user.save(on: req).map(to: Response.self) { user in
        // 5
        try req.authenticateSession(user)
        // 6
        return req.redirect(to: "/")
    }
}
```

这是路由处理程序中发生的事情：

1. 定义路由处理程序，它接受请求和解码的RegisterData。
2. 哈希表单提交的密码。
3. 使用表单中的数据和哈希密码创建新用户。
4. 保存新用户并解包返回的future。
5. 验证新用户的session。这会在用户注册时自动登录，从而在注册网站时提供良好的用户体验。
6. 将重定向返回主页。

接下来，在boot(router:)中的authSessionRoutes.post("logout",use: logoutHandler)下面添加以下内容：

```
// 1
authSessionRoutes.get("register", use: registerHandler)
// 2
authSessionRoutes.post(RegisterData.self, at: "register",
    use: registerPostHandler)
```

这是它的作用：

1. 将/**register**的GET请求连接到registerHandler(_:)。
2. 将/**register**的POST请求连接到registerPostHandler(_:data:)。将请求的body解码为RegisterData。

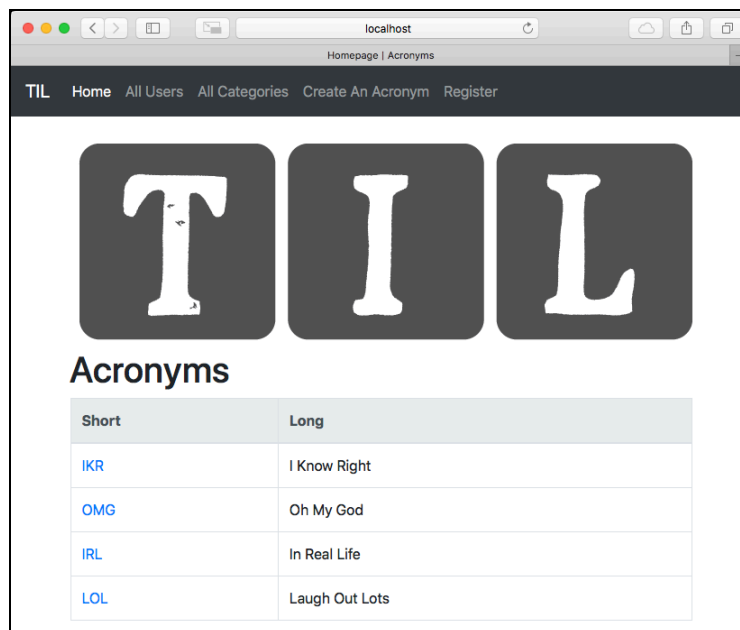
最后，打开base.leaf。在导航栏中的结束标签之前，添加以下内容：

```
/// 1
#if(!userLoggedIn) {
  /// 2
  <li class="nav-item #if(title == "Register"){active}">
    /// 3
    <a href="/register" class="nav-link">Register</a>
  </li>
}
```

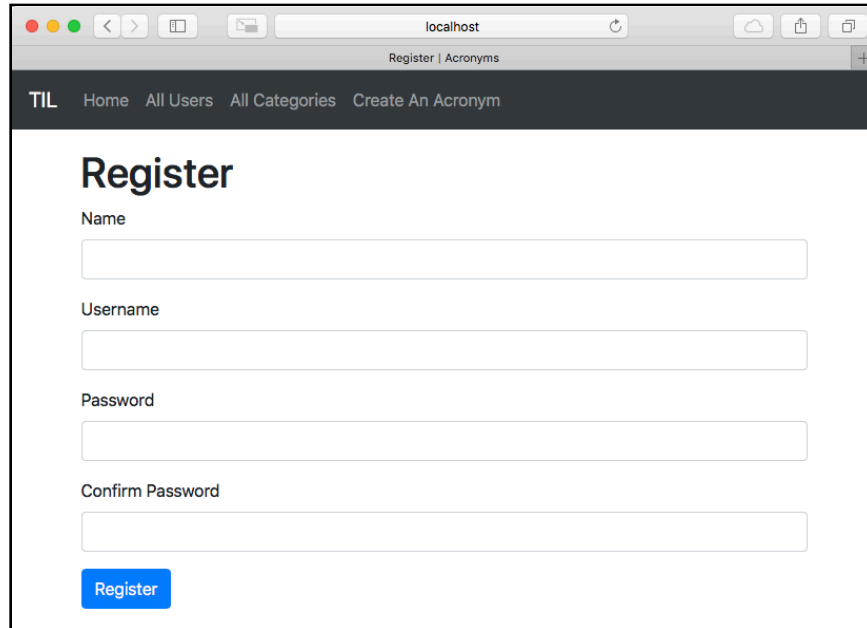
这是新Leaf代码的作用：

1. 检查用户是否已登录。如果没有登录，您只想显示注册链接。
2. 将新导航链接添加到导航栏。如果当前页面是“**Register**”页面，请设置active类。
3. 添加指向/**register**新路由的链接。

保存模板，然后在Xcode中构建并运行项目。在浏览器中访问<http://localhost:8080>。您将看到新的导航链接：



单击“**Register**”，您将看到新的注册页面：



如果您填写表单并单击“**Register**”，该应用程序将带您进入主页。注意右上角的“**Log out**”按钮；这证实了注册会自动将您登录。

基本验证

Vapor提供一个验证模块，帮助您检查数据和模型。打开**WebsiteController**，并在底部添加以下内容：

```
// 1
extension RegisterData: Validatable, Reflectable {
// 2
    static func validations() throws
        -> Validations<RegisterData> {
// 3
        var validations = Validations(RegisterData.self)
// 4
        try validations.add(\.name, .ascii)
// 5
        try validations.add(\.username,
            .alphanumeric && .count(3...))
// 6
        try validations.add(\.password, .count(8...))
// 7
        return validations
    }
}
```

这是它的作用：

1. 扩展RegisterData以使其遵循Validatable和Reflectable协议。Validatable允许您使用Vapor验证类型。 Reflectable提供了一种发现类型内部组件的方法。
2. 根据Validatable的要求实现validations()。
3. 创建一个Validations实例以包含各种验证器。
4. 添加验证器以确保RegisterData的名称仅包含ASCII字符。
注意：在对此类名称添加限制时要小心。某些国家/地区（如中国）没有ASCII字符的名称。
5. 添加验证器以确保用户名仅包含字母数字字符且长度至少为3个字符。 `.count(_)`采用Swift的区间运算符，允许您根据需要创建半开区间和闭区间。
6. 添加验证器以确保密码长度至少为8个字符。
7. 返回Vapor验证以测试。

如您所见，Vapor允许您在模型或传入数据上创建强大的验证。因为您使用key paths，Vapor会创建类型安全的验证。请注意`.ascii`验证程序仅适用于String类型。例如，它不适用于Int。

在`registerPostHandler(_ :data:)`中，在方法的顶部添加以下内容：

```
do {  
    try data.validate()  
} catch {  
    return req.future(req.redirect(to: "/register"))  
}
```

这会对已解码的RegisterData调用`validate()`，检查先前添加的每个验证器。 `validate()`可以抛出`ValidationError`。在API中，您可以将此错误传播回用户，但在网站上，这不会带来良好的用户体验。在这种情况下，您应该将用户重定向回“register”页面。

构建并运行，然后访问浏览器中的“register”页面。如果您输入的信息与验证器不符，则应用程序会将您送回重试。

自定义验证

如果您一直密切关注，您会发现验证中存在缺陷：没有什么可以确保密码匹配！ Vapor 的验证库没有提供检查两个字符串匹配的内置方法。但是，添加自定义验证器很容易。在 `RegisterData` 的 `validations()` 中，在返回验证之前，添加以下内容：

```
// 1
validations.add("passwords match") { model in
// 2
    guard model.password == model.confirmPassword else {
// 3
        throw BasicValidationError("passwords don't match")
    }
}
```

以下是新验证器的作用：

1. 使用 `Validation` 的 `add (_:_:)` 为 `RegisterData` 添加自定义验证器。这将可读描述作为第一个参数。第二个参数是一个闭包，如果验证失败则应该抛出。
2. 验证 `password` 和 `confirmPassword` 是否匹配。
3. 如果他们不匹配，抛出 `BasicValidationError`。

构建并运行，然后尝试注册用户并且密码不匹配。该应用程序会将您重定向回“register”表单。

显示错误

目前，当用户错误地填写表单时，应用程序会重定向回表单，而不会显示任何错误。打开 `register.leaf` 并在 `<h1>#(title)</h1>` 下面添加以下内容：

```
#if(message) {
    <div class="alert alert-danger" role="alert">
        Please fix the following errors:<br />
        #(message)
    </div>
}
```

如果页面上下文包含message，则会将其显示在新的<div>中。您可以通过设置alert和alert-danger类来适当地设置新消息的样式。打开**WebsiteController.swift**，并将以下内容添加到RegisterContext的末尾：

```
let message: String?

init(message: String? = nil) {
    self.message = message
}
```

这是要在注册页面上显示的消息。请记住，Leaf会优雅地处理nil，允许您在正常情况下使用默认值。

这是模板使用的标志。在registerHandler(:)中，替换：

```
let context = RegisterContext()
```

用以下内容：

```
let context: RegisterContext
if let message = req.query[String.self, at: "message"] {
    context = RegisterContext(message: message)
} else {
    context = RegisterContext()
}
```

这会检查请求的query。如果message存在 - 即URL是/register?

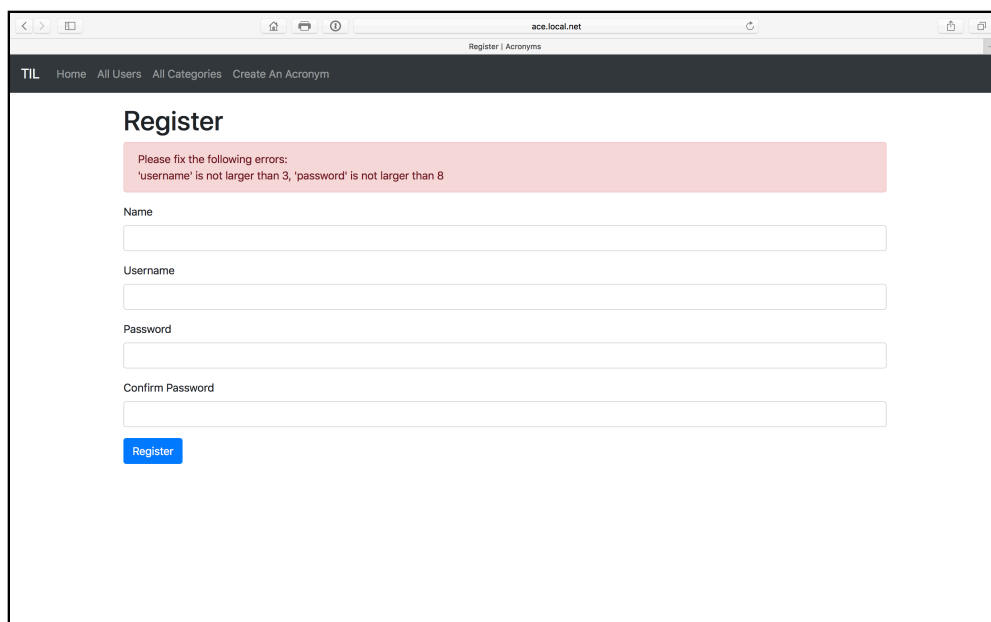
message=some-string - 路由处理程序将其包含在Leaf用于渲染页面的上下文中。

最后，在registerPostHandler(:data:)中，将catch闭包替换为：

```
catch (let error) {
    let redirect: String
    if let error = error as? ValidationError,
        let message = error.reason.addingPercentEncoding(
            withAllowedCharacters: .urlQueryAllowed) {
        redirect = "/register?message=\(message)"
    } else {
        redirect = "/register?message=Unknown+error"
    }
    return req.future(req.redirect(to: redirect))
}
```

验证失败时，路由处理程序从ValidationError中提取消息，将其正确转义以包含在URL中，并将其添加到重定向URL。然后，它将用户重定向回注册页面。构建并运行，然后在浏览器中访问<http://localhost:8080/register>。

提交空表单，您将看到新消息：



The screenshot shows a web browser window with the address bar displaying 'ace.local.net'. The page title is 'Register | Acronyms'. The navigation bar includes links: 'TIL', 'Home', 'All Users', 'All Categories', and 'Create An Acronym'. The main content area is titled 'Register'. Below the title, a red error message box states: 'Please fix the following errors: 'username' is not larger than 3, 'password' is not larger than 8'. The form contains four input fields: 'Name', 'Username', 'Password', and 'Confirm Password'. A blue 'Register' button is located at the bottom of the form.

然后去哪儿？

在本章中，您学习了如何使用Vapor的验证库来检查请求的数据。您也可以对模型和其他类型进行验证。

在下一章中，您将学习如何将TIL应用程序与OAuth提供商程序集成。这使您可以将登录和注册委派给Google或GitHub等在线服务，从而允许用户使用现有帐户登录。