Chapter 25: Adding Profile Pictures

By Tim Condon

在前面的章节中，您学习了如何在POST请求中将数据发送到Vapor应用程序。您使用了JSON报文和表单来传输数据，但数据始终是简单文本。在本章中，您将学习如何在请求中发送文件并在Vapor应用程序中处理这些文件。您将使用此知识允许用户在Web应用程序中上传个人资料图片。

> 注意：本章将教您如何将文件上传到运行Vapor应用程序的服务器。对于实际应用程序，您应该考虑将文件转发到存储服务器，例如AWS S3。许多托管服务提供商（如Vapor Cloud和Heroku）不提供持久化存储。这意味着在重新部署应用程序时，您将丢失上传的文件。如果托管服务提供商重新启动您的应用程序，您也将丢失文件。此外，将文件上传到同一服务器意味着您无法将应用程序扩展到多个实例，因为这些文件没有存在于所有应用程序实例中。

将图片添加到模型中

与前面的章节一样，您需要更改模型，以便将图像与用户关联起来。在Xcode中打开Vapor TIL应用程序，并打开**User.swift**。在var email: String下面添加以下内容：

```
var profilePicture: String?
```

这会为图像存储一个可选的字符串。它将包含磁盘上用户个人图片的文件名。文件名是可选的，因为您没有强制要求用户具有个人图片 - 他们在注册时没有个人图片。替换初始化程序以支持新属性：

```
init(name: String,
     username: String,
     password: String,
     email: String,
     profilePicture: String? = nil) {
  self.name = name
  self.username = username
  self.password = password
  self.email = email
  self.profilePicture = profilePicture
}
```

为profilePicture提供默认值nil允许您的应用程序继续编译和操作，无需进一步更改源代码。

> 注意：您可以使用Google和GitHub中的用户API获取用户个人图片的网址。这将允许您下载图像并将其与用户的常规图片一起存储或保存链接。但是，这将留给读者作为练习。

您可以将上传个人图片作为注册体验的一部分，但本章将在单独的步骤中完成。请注意UsersController中的createHandler(_:user:)不需要为新属性进行更改。这是因为路由处理程序使用Codable，并且如果POST请求中不存在数据，则将该属性设置为nil。

重置数据库

与过去一样，由于您已向User添加了属性，因此必须重置数据库。不像第24章“Password Reset & Emails”中那样删除Docker容器，本章使用revert命令。**Option** + 单击Xcode中的“**Run**”按钮以打开方案编辑器。在**Arguments**选项卡上，在**Arcuments Passed On Launch**部分单击“**+**”。输入：

```
revert --all --yes
```

单击“**Run**”，您将在Xcode控制台中看到显示还原的输出。
Option + 再次单击“**Run**”按钮，然后清除您输入的参数旁边的复选框。下次应用程序启动时，它将使用新列准备数据库。

> 注意：还原数据库或重置Docker容器的结果没有区别。无论您选择哪一个仅取决于您个人喜好。

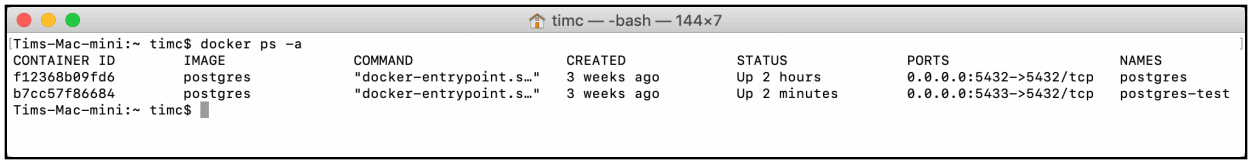
验证测试

由于您更改了用户模型，因此应运行测试程序以确保更改没有破坏任何内容。在Xcode中，选择**TILApp-Package**方案。接下来，确保测试数据库的Docker容器正在运行。在终端中，键入：

```
docker ps -a
```

您应该看到主数据库容器**postgres**，和测试数据库容器**postgres-test**。确保**postgres-test**的状态类似于“**Up 2 hours**”。如果状态为“**Exited**”，则可以使用以下命令再次启动容器：

```
docker start postgres-test
```



最后，在Xcode中，Option + 单击**TILApp-Package**方案并选择**Test**操作。确保取消选中“**Use the Run action's arguments and environment variables**”。然后，确保已设置了所需的环境变量。它们可以是为测试的随机值：

- **GOOGLE_CALLBACK_URL**
- **GOOGLE_CLIENT_ID**
- **GOOGLE_CLIENT_SECRET**
- **GITHUB_CALLBACK_URL**
- **GITHUB_CLIENT_ID**
- **GITHUB_CLIENT_SECRET**
- **SENDGRID_API_KEY**

单击“Close”，然后键入Command + U以运行所有测试。他们应该都能通过。

创建表单

随着模型的更改， 您现在可以创建一个页面以允许用户提交图片。 在Xcode中， 打开**WebsiteController.swift**。在resetPasswordPostHandler(_:data:)下面添加以下内容：

```
func addProfilePictureHandler(_ req: Request) throws
  -> Future<View> {
    return try req.parameters.next(User.self)
      .flatMap { user in
        try req.view().render(
          "addProfilePicture",
          ["title": "Add Profile Picture",
           "username": user.name])
   }
}
```

这定义了一个渲染**addProfilePicture.leaf**的新路由处理程序。路由处理程序还将标题和用户名称作为字典传递给模板。接下来，将以下内容添加到boot(router:)的末尾，以注册新的路由处理程序：

```
protectedRoutes.get(
  "users",
  User.parameter,
  "addProfilePicture",
  use: addProfilePictureHandler)
```

这将**/users/<USER_ID>/addProfilePicture** 的GET请求连接到addProfilePictureHandler(_:)。请注意，该路由也是受保护的路由 - 用户必须登录才能添加个人图片。 TIL应用程序还允许用户为任何用户上传个人图片，而不仅仅是他们自己的。

在**Resources/Views**中，创建新模板**addProfilePicture.leaf**。在编辑器中打开新文件并插入以下内容：

```
#// 1
#set("content") {
  #// 2
  <h1>#(title)</h1>

  #// 3
  <form method="post" enctype="multipart/form-data">
    #// 4
    <div class="form-group">
      <label for="picture">
        Select Picture for #(username)
      </label>
      <input type="file" name="picture"
       class="form-control-file" id="picture"/>
    </div>
```

```
    #// 5
    <button type="submit" class="btn btn-primary">
      Upload
    </button>
  </form>
}

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

这是新模板的作用：

1. 根据**base.leaf**的要求设置content。

2. 使用传递给模板的标题作为页面的标题。

3. 创建一个表单并将方法设置为POST。提交表单时，浏览器将表单作为POST请求发送到同一URL。注意multipart/form-data的编码类型。这允许您从浏览器将文件发送到服务器。

4. 创建一个表单组，input类型为file。这将在您的Web浏览器中显示文件浏览器。Bootstrap使用form-control-file来帮助设置input样式。

5. 添加提交按钮以允许用户提交表单。

6. 嵌入**base.leaf**以包含主模板。

接下来，您需要一个链接，供用户访问新表单。打开**WebsiteController.swift**，在UserContext的底部添加一个新属性：

```
let authenticatedUser: User?
```

这将存储该请求的已认证的用户（如果存在）。在userHandler(_:)中，使用以下内容替换let context = ...：

```
// 1
let loggedInUser = try req.authenticated(User.self)
// 2
let context = UserContext(
  title: user.name,
  user: user,
  acronyms: acronyms,
  authenticatedUser: loggedInUser)
```

这是你改变的：

1. 从请求中获取已认证的用户。这会返回User?，因为可能没有已认证的用户。

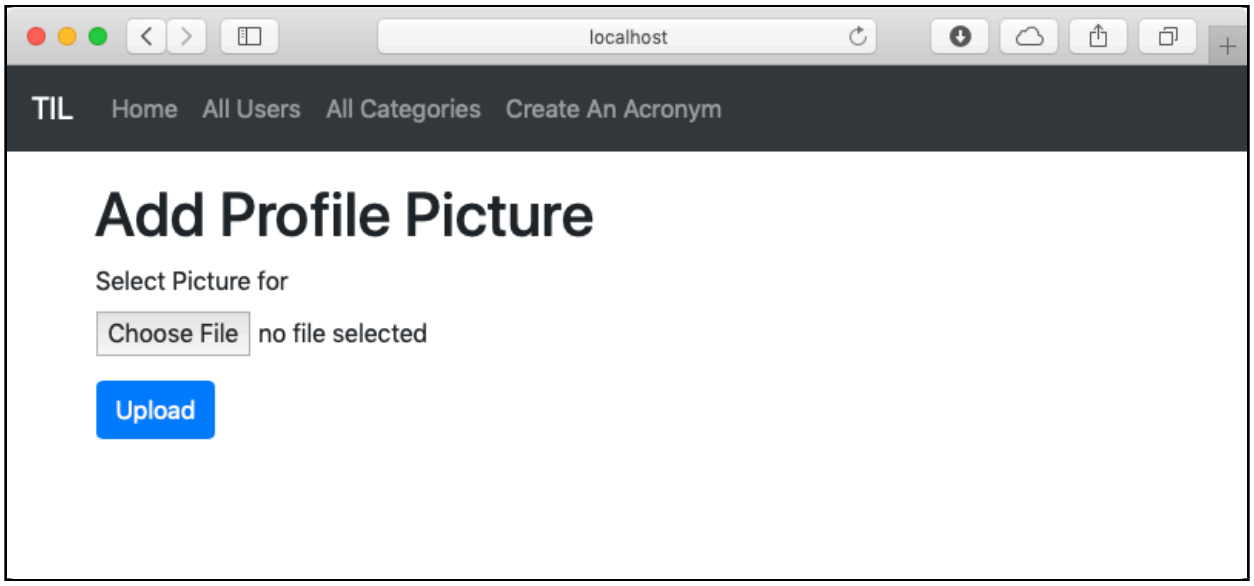
2. 将可选的已认证的用户传递给上下文。

最后，打开**user.leaf**。在#embed("acronymsTable")之前添加以下内容：

```
#if(authenticatedUser) {
  <a href="/users/#(user.id)/addProfilePicture">
    #if(user.profilePicture){Update } else{Add } Profile Picture
  </a>
}
```

如果用户已登录，则会添加链接以指向新添个人图片页面。如果用户已有个人图片，则链接将显示更新个人图片，否则链接将显示添加个人图片。

在Xcode中，选择Run方案，构建并运行应用程序。在浏览器中，转到**http://localhost:8080/login** 并以**admin**用户身份登录。登录后，单击“**All Users**”并选择**admin**用户。在添加个人图片页面有一个新链接。单击**Add Profile Picture**，您将看到要添加个人图片的新表单：



接受文件上传

接下来，实现必要的代码来处理来自表单的**POST**请求。在终端中，输入以下内容：

```
# 1
mkdir ProfilePictures
# 2
touch ProfilePictures/.keep
```

以下是这些命令的作用：

1. 创建用于存储用户个人图片的目录。

2. 添加一个空文件，以便将该目录添加到源代码管理中。这有助于部署应用程序以确保目录存在。

接下来，在Xcode中，打开**WebsiteController.swift**。在文件的底部添加以下内容：

```
struct ImageUploadData: Content {
  var picture: Data
}
```

此新类型表示表单发送的数据。 picture匹配HTML表单中指定的input的名称。

因为表单上传的是文件，所以您要将图片解码为Data。

接下来，在WebsiteController的顶部添加一个新属性，在boot(router:)之上：

```
let imageFolder = "ProfilePictures/"
```

这将定义您存储图像的文件夹。接下来，在addProfilePictureHandler(_:)的下面为**POST**请求添加请求处理程序：

```
func addProfilePicturePostHandler(_ req: Request)
  throws -> Future<Response> {
    // 1
    return try flatMap(
      to: Response.self,
      req.parameters.next(User.self),
      req.content.decode(ImageUploadData.self)) {
        user, imageData in
        // 2
        let workPath =
          try req.make(DirectoryConfig.self).workDir
        // 3
        let name =
          try "\(user.requireID())-\(UUID().uuidString).jpg"
        // 4
        let path = workPath + self.imageFolder + name
        // 5
        FileManager().createFile(
          atPath: path,
          contents: imageData.picture,
          attributes: nil)
        // 6
        user.profilePicture = name
        // 7
        let redirect =
          try req.redirect(to: "/users/\(user.requireID())")
        return user.save(on: req).transform(to: redirect)
```

```
    }
}
```

以下是新请求处理程序的作用：

1. 从参数中获取用户并将请求body解码为ImageUploadData。
2. 获取应用程序的当前工作目录。
3. 为个人图片创建唯一名称。
4. 设置要保存的文件的路径。
5. 使用路径和图像数据将文件保存在磁盘上。
6. 使用个人图片文件名更新用户。
7. 保存已更新的用户，并返回重定向到用户页面。

最后，在boot(router:)底部注册路由：

```
protectedRoutes.post(
  "users",
  User.parameter,
  "addProfilePicture",
  use: addProfilePicturePostHandler)
```

这将**/users/<USER_ID>/addProfilePicture**的**POST**请求连接到addProfilePicturePostHandler(_:)。

显示图片

现在，用户可以上传个人照片，您需要能够将图像提供回浏览器。通常，您将使用FileMiddleware中间件。但是，当您将图像存储在不同的目录中时，本章将教您如何手动提供这些图像。

在**WebsiteController.swift**中，在addProfilePicturePostHandler(_:)下面添加一个新的路由处理程序：

```
func getUsersProfilePictureHandler(_ req: Request)
  throws -> Future<Response> {
    // 1
    return try req.parameters.next(User.self)
      .flatMap(to: Response.self) { user in
        // 2
        guard let filename = user.profilePicture else {
          throw Abort(.notFound)
```

```
        }
        // 3
        let path = try req.make(DirectoryConfig.self)
          .workDir + self.imageFolder + filename
      // 4
      return try req.streamFile(at: path)
    }
}
```

这是新路由处理程序的作用：

1. 从请求的参数中获取用户。
2. 确保用户已保存个人图片，否则抛出**404 Not Found**错误。
3. 构造用户个人图片的路径
4. 使用Vapor的FileIO方法将文件作为响应返回。这会处理读取文件并将正确的信息返回给浏览器。

接下来，在boot(router:)的authSessionRoutes.post(ResetPasswordData.self,at:"resetPassword",use:resetPasswordPostHandler)下面注册新路由：

```
authSessionRoutes.get(
  "users",
  User.parameter,
  "profilePicture",
  use: getUsersProfilePictureHandler)
```

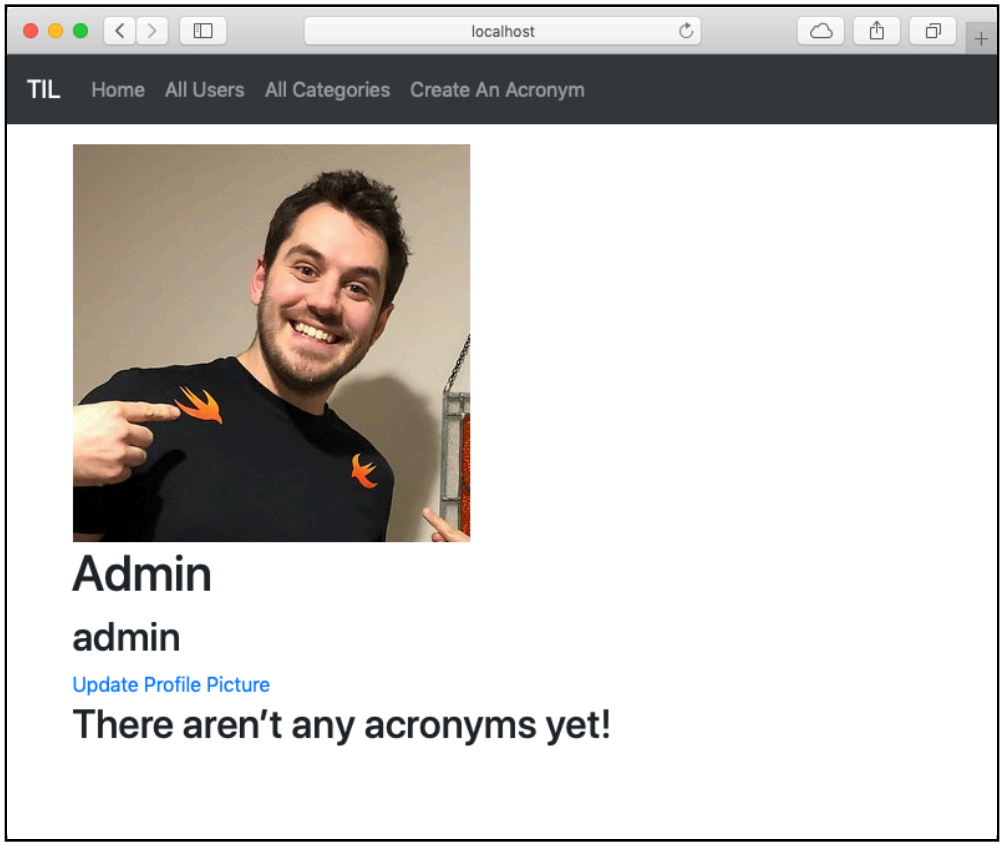
这将**/users/<USER_ID>/profilePicture**的**GET**请求连接到getUsersProfilePictureHandler(_:)。最后，打开**user.leaf**。在<h1>#(user.name)</h1>之前添加以下内容：

```
#if(user.profilePicture) {
  <img src="/users/#(user.id)/profilePicture"
   alt="#(user.name)">
}
```

这将检查传递给模板的上下文的用户是否有个人图片。如果有，Leaf会将图像链接添加到页面。

构建并运行应用程序，然后在浏览器中转到**http://localhost:8080/login**。以默认管理员用户身份登录，然后导航到管理员用户的个人资料页面。单击**Add Profile Picture**，然后在表单中单击**Choose File**。选择要上传的图像，然后单击**Upload**。

该网站会将您重定向到用户的个人资料页面，您可以在其中看到上传的图片：



然后去哪儿?

在本章中，您学习了如何处理Vapor中的文件。您了解了如何处理文件上传并将文件保存到磁盘。您还学习了如何在路由处理程序中从磁盘提供文件服务。

您现在已经构建了一个功能齐全的API，它展示了Vapor的许多功能。您已经构建了一个iOS应用程序来使用API，以及使用Leaf的前端网站。您还学习了如何测试您的应用程序。

这些章节已经为您提供了构建您自己的应用程序的后端和网站所需的所有知识！接下来的章节将介绍您可能需要的更多高级话题，例如数据库migrations和缓存。您还将学习如何将应用程序部署到Internet。