Chapter 13: Creating a Simple iPhone App, Part 2

By Tim Condon

在上一章中，您创建了一个可以创建用户和缩略词的iPhone应用程序。在本章中，您将扩展应用程序以包括查看有关单个缩略词的详细信息。您还将学习如何执行最终的CRUD操作：编辑和删除。最后，您将学习如何向类别添加缩略词。

> 注意：本章希望您运行TIL Vapor应用程序。它还希望您已完成上一章中的iOS应用程序。如果没有，获取starter项目并从那里拿起。有关如何运行Vapor应用程序的详细信息，请参见第12章“Creating a Simple iPhone App Part 1”。

Getting started

在上一章中，您学习了如何查看表中的所有缩略词。现在，您希望在用户点击列表单元格时显示有关单个缩略词的所有信息。starter项目包含了必要的内容；你只需要实现详细信息。

打开**AcronymsTableViewController.swift**。在viewWillAppear(_:)之后添加以下内容：

```
// MARK: - Navigation
override func prepare(
  for segue: UIStoryboardSegue,
  sender: Any?
) {
  // 1
  if segue.identifier == "AcronymsToAcronymDetail" {
    // 2
    guard
```

```
        let destination =
          segue.destination as? AcronymDetailTableViewController,
        let indexPath = tableView.indexPathForSelectedRow
        else {
          return
      }

      // 3
      destination.acronym = acronyms[indexPath.row]
    }
  }
```

这是它的作用：

1. 验证预期的segue。
2. 确保目标视图控制器是AcronymDetailTableViewController。
3. 将AcronymDetailTableViewController中的缩略词属性设置为所选的缩略词。

在**Utilities**组中创建一个名为**AcronymRequest.swift**的新Swift文件。打开新文件并为缩略词的用户请求添加枚举：

```
enum AcronymUserRequestResult {
  case success(User)
  case failure
}
```

这定义了存储创建用户的成功情况和失败情况。由于类别由数组表示，因此缩略词的类别请求将使用现有的GetResourcesRequest。在文件的底部创建一个新类型来表示缩略词资源请求：

```
struct AcronymRequest {
  let resource: URL

  init(acronymID: Int) {
    let resourceString =
      "http://localhost:8080/api/acronyms/\(acronymID)"
    guard let resourceURL = URL(string: resourceString) else {
      fatalError()
    }
    self.resource = resourceURL
  }
}
```

这会将resource属性设置为缩略词的URL。

在AcronymRequest的底部添加一个方法来获取缩略词的用户：

```
func getUser(
  completion: @escaping (AcronymUserRequestResult) -> Void) {

  // 1
  let url = resource.appendingPathComponent("user")

  // 2
  let dataTask = URLSession.shared
    .dataTask(with: url) { data, _, _ in
      // 3
      guard let jsonData = data else {
        completion(.failure)
        return
      }
      do {
      // 4
        let user = try JSONDecoder()
          .decode(User.self, from: jsonData)
        completion(.success(user))
      } catch {
        // 5
        completion(.failure)
      }
  }
  // 6
  dataTask.resume()
}
```

这是它的作用：

1. 创建URL以获取缩略词的用户。
2. 使用共享URLSession创建数据任务。
3. 检查响应是否包含body，否则失败。
4. 将响应body解码为User对象，并使用成功结果调用完成处理程序。
5. 捕获任何解码错误并使用失败结果调用完成处理程序。
6. 启动网络任务。

在getUser(completion:)下面添加以下方法来获取用户的类别：

```
func getCategories(
  completion: @escaping (GetResourcesRequest<Category>) -> Void
) {
  let url = resource.appendingPathComponent("categories")
  let dataTask = URLSession.shared
    .dataTask(with: url) { data, _, _ in
      guard let jsonData = data else {
```

```
        completion(.failure)
        return
      }
      do {
      let categories = try JSONDecoder()
        .decode([Category].self, from: jsonData)
      completion(.success(categories))
      } catch {
        completion(.failure)
      }
  }
  dataTask.resume()
}
```

这与项目中的其他请求方法完全相同，将响应body解码为[Category]。

打开**AcronymDetailTableViewController.swift**并将以下实现添加到getAcronymData()：

```
// 1
guard let id = acronym?.id else {
  return
}

// 2
let acronymDetailRequester = AcronymRequest(acronymID: id)
// 3
acronymDetailRequester.getUser { [weak self] result in
  switch result {
  case .success(let user):
    self?.user = user
  case .failure:
    let message =
      "There was an error getting the acronym's user"
    ErrorPresenter.showError(message: message, on: self)
  }
}

// 4
acronymDetailRequester.getCategories { [weak self] result in
  switch result {
  case .success(let categories):
    self?.categories = categories
  case .failure:
    let message =
      "There was an error getting the acronym's categories"
    ErrorPresenter.showError(message: message, on: self)
  }
}
```

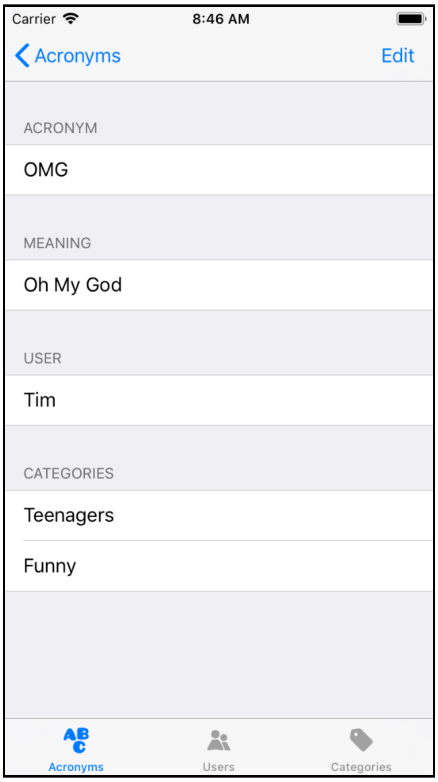
这是它的作用：

1. 确保缩略词ID不为nil。
2. 创建AcronymRequest以收集信息。
3. 获取缩略词的用户。如果请求成功，更新user属性。否则，显示相应的错误消息。
4. 获取缩略词的类别。如果请求成功，更新categories属性。否则，显示相应的错误消息。

该项目在包含四个sections的列表视图中显示缩略词数据。这些是：

- 缩略词
- 它的含义
- 它的创建用户
- 它的类别

构建并运行。点击Acronyms列表中的缩略词，应用程序将显示包含所有信息的详细信息视图：



编辑缩略词

要编辑缩略词，用户可以点击缩略词详细信息视图中的“**Edit**”按钮。打开**CreateAcronymTableViewController.swift**。acronym属性用于存储当前的缩略词。如果设置了此属性 - 通过在**AcronymDetailTableViewController.swift**中的prepare(for:sender:) - 则用户正在编辑缩略词。否则，用户正在创建一个新的缩略词。

在viewDidLoad()中，将populateUsers()替换为：

```
if let acronym = acronym {
  acronymShortTextField.text = acronym.short
  acronymLongTextField.text = acronym.long
  userLabel.text = selectedUser?.name
  navigationItem.title = "Edit Acronym"
} else {
  populateUsers()
}
```

如果设置了缩略词，则表示您处于编辑模式。使用正确的值填充显示字段并更新视图的标题。如果您处于创建模式，像之前一样调用populateUsers()。

要更新缩略词，请对API中的缩略词资源发出PUT请求。打开**AcronymRequest.swift**并在AcronymRequest底部添加一个方法来更新缩略词：

```
func update(
  with updateData: Acronym,
  completion: @escaping (SaveResult<Acronym>) -> Void
) {
  do {
    // 1
    var urlRequest = URLRequest(url: resource)
    urlRequest.httpMethod = "PUT"
    urlRequest.httpBody = try JSONEncoder().encode(updateData)
    urlRequest.addValue("application/json",
                        forHTTPHeaderField: "Content-Type")
    let dataTask = URLSession.shared
      .dataTask(with: urlRequest) { data, response, _ in
        // 2
        guard
          let httpResponse = response as? HTTPURLResponse,
          httpResponse.statusCode == 200,
          let jsonData = data
          else {
            completion(.failure)
            return
        }
        do {
        // 3
```

```
            let acronym = try JSONDecoder()
              .decode(Acronym.self, from: jsonData)
            completion(.success(acronym))
          } catch {
            completion(.failure)
          }
      }
      dataTask.resume()
    } catch {
      completion(.failure)
    }
  }
```

此方法与您构建的其他请求一样。不同之处是：

1. 创建和配置URLRequest。该方法必须是**PUT**，并且请求body包含已编码的缩略词数据。设置正确的标头，以便Vapor应用程序知道请求包含JSON。

2. 确保响应是HTTP响应，状态码为200，响应具有body。

3. 将响应body解码为Acronym并使用成功结果调用完成处理程序。

返回到**CreateAcronymTableViewController.swift**。在save(_:)里，用以下内容替换函数在 let acronym = Acronym(short: shortText, long: longText, userID: userID)之后的其余部分。

```
  if self.acronym != nil {
    // update code goes here
  } else {
    ResourceRequest<Acronym>(resourcePath: "acronyms")
      .save(acronym) { [weak self] result in
        switch result {
        case .failure:
          let message = "There was a problem saving the acronym"
          ErrorPresenter.showError(message: message, on: self)
        case .success:
          DispatchQueue.main.async { [weak self] in
            self?.navigationController?
              .popViewController(animated: true)
          }
        }
    }
  }
```

这将检查类的acronym属性以查看它是否已设置。如果属性为nil，则用户正在保存新的缩略词，因此该函数将执行与之前相同的保存请求。

在if块内的// update code goes here注释之后，添加以下代码以更新缩略词：

```
// 1
guard let existingID = self.acronym?.id else {
  let message = "There was an error updating the acronym"
  ErrorPresenter.showError(message: message, on: self)
  return
}
// 2
AcronymRequest(acronymID: existingID)
  .update(with: acronym) { result in
    switch result {
    // 3
    case .failure:
      let message = "There was a problem saving the acronym"
      ErrorPresenter.showError(message: message, on: self)
    case .success(let updatedAcronym):
      self.acronym = updatedAcronym
      DispatchQueue.main.async { [weak self] in
        // 4
        self?.performSegue(
          withIdentifier: "UpdateAcronymDetails",
          sender: nil)
      }
    }
}
```

以下是更新代码的作用：

1. 确保缩略词具有有效ID。

2. 创建一个AcronymRequest并调用update(with:completion:)。

3. 如果更新失败，则显示错误消息。

4. 如果更新成功，则存储更新的缩略词并触发unwind segue跳转到AcronymsDetailTableViewController。

接下来，打开**AcronymsDetailTableViewController.swift**并将以下实现添加到prepare(for:sender:)的末尾：

```
if segue.identifier == "EditAcronymSegue" {
  // 1.
  guard
    let destination = segue.destination
      as? CreateAcronymTableViewController else {
    return
  }

  // 2.
  destination.selectedUser = user
  destination.acronym = acronym
}
```

这是它的作用：

1. 确保目标控制器是CreateAcronymTableViewController。
2. 设置在目标控制器上selectedUser和acronym属性。

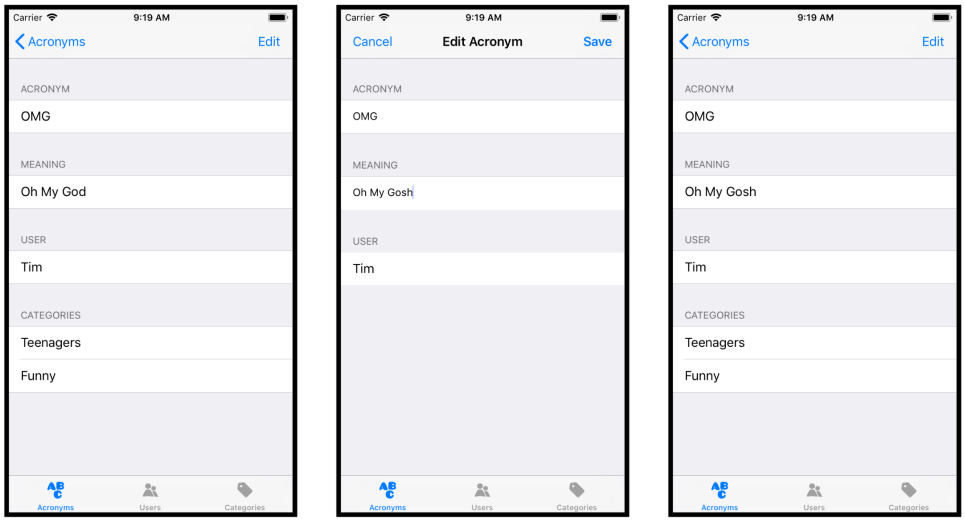
接下来，将以下实现添加到unwind segue的目标updateAcronymDetails(_:)：

```
guard let controller = segue.source
  as? CreateAcronymTableViewController else {
  return
}

user = controller.selectedUser
acronym = controller.acronym
```

这将捕获更新的缩略词和用户，从而触发对其自身视图的更新。

构建并运行。点击缩略词以打开缩写词详细信息视图，然后点按“**Edit**”。更改详细信息，然后点按“**Save**”。该视图将返回到已更新值的缩略词详细信息页面：



删除缩略词

打开**AcronymRequest.swift**并添加一个方法来删除最后的缩略词：

```
func delete() {
  // 1
  var urlRequest = URLRequest(url: resource)
  urlRequest.httpMethod = "DELETE"
  // 2
```

```
    let dataTask = URLSession.shared.dataTask(with: urlRequest)
    dataTask.resume()
  }
```

这是delete()的作用：

1. 创建URLRequest并将HTTP方法设置为**DELETE**。

2. 使用共享URLSession为请求创建数据任务并发送请求。这会忽略请求的结果。

打开**AcronymsTableViewController.swift**。要启用列表行删除功能，请在tableView(_:cellForRowAt:)之后添加以下内容：

```
  override func tableView(
    _ tableView: UITableView,
    commit editingStyle: UITableViewCell.EditingStyle,
    forRowAt indexPath: IndexPath
  ) {
    if let id = acronyms[indexPath.row].id {
      // 1
      let acronymDetailRequester = AcronymRequest(acronymID: id)
      acronymDetailRequester.delete()
    }

    // 2
    acronyms.remove(at: indexPath.row)
    // 3
    tableView.deleteRows(at: [indexPath], with: .automatic)
  }
```

这样可以在列表视图上启用“滑动删除”功能。以下是它的工作原理：

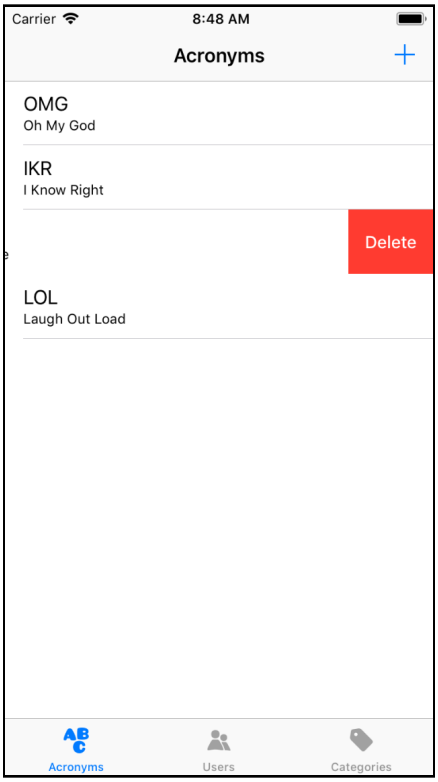
1. 如果缩略词具有有效ID，就为缩略词创建AcronymRequest，并调用delete()以删除API中的缩略词。

2. 从本地的缩略词数组中删除缩略词。

3. 从列表视图中删除缩略词的行。

构建并运行。在缩略词上向左滑动，将出现**Delete**按钮。点击**Delete**以删除缩略词。

如果您对列表视图进行下拉刷新，该缩略词不再出现了，因为该应用程序已在API中将其删除：



创建类别

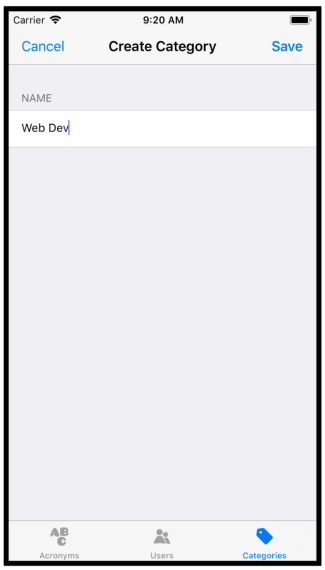
建立创建类别列表就像建立创建用户列表一样。打开**CreateCategoryTableViewController.swift**并将save(_:)的实现替换为：

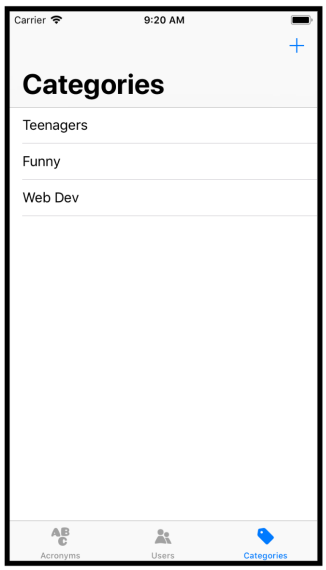
```
// 1
guard
  let name = nameTextField.text,
  !name.isEmpty
  else {
    ErrorPresenter.showError(
      message: "You must specify a name", on: self)
    return
}

// 2
let category = Category(name: name)
// 3
ResourceRequest<Category>(resourcePath: "categories")
  .save(category) { [weak self] result in
    switch result {
    // 5
```

```
      case .failure:
        let message = "There was a problem saving the category"
        ErrorPresenter.showError(message: message, on: self)
      // 6
      case .success:
        DispatchQueue.main.async { [weak self] in
          self?.navigationController?
            .popViewController(animated: true)
        }
      }
  }
```

这就像保存用户的save(_:)方法一样。构建并运行。在“Categories”选项卡上，点击“+”按钮以打开“**Create Category**”视图。填写类别名称，然后点按“**Save**”。如果保存成功，本页面将关闭，新类别将显示在列表中：





缩略词加入到类别

完成后，您必须实现缩略词加入到类别的功能。添加一个新的section列表行到缩略词详细信息视图，其中包含用于将缩略词加入到类别的按钮。

打开**AcronymsDetailTableViewController.swift**。将numberOfSections(in:)中的return语句更改为：

```
  return 5
```

在tableView(_:cellForRowAt:)中，在switch语句的default之前添加一个新的情况：

```
// 1
case 4:
  cell.textLabel?.text = "Add To Category"
```

接着，就在return cell之前添加以下内容：

```
// 2
if indexPath.section == 4 {
  cell.selectionStyle = .default
  cell.isUserInteractionEnabled = true
} else {
  cell.selectionStyle = .none
  cell.isUserInteractionEnabled = false
}
```

这些步骤：

1. 如果单元格位于新的section里，则将列表单元格标题设置为“Add To Category”。

2. 如果单元格在新的section中，则在单元格上启用选择，否则禁用选择。这允许用户选择新行而不选择其他行。

starter项目已包含此新列表的视图控制器：**AddToCategoryTableViewController.swift**。该类定义了三个关键属性：

- `categories`：从API检索的所有类别的数组。
- `selectedCategories`：为缩略词选择的类别。
- `acronym`：加入到类别的缩略词。

该类还包含UITableViewDataSource的扩展方法。如果类别在selectedCategories数组中，则tableView(_:cellForRowAt:)在单元格上设置accessoryType。

打开**AddToCategoryTableViewController.swift**并将以下实现添加到loadData()以获取API中的所有类别：

```
// 1
let categoriesRequest =
  ResourceRequest<Category>(resourcePath: "categories")
// 2
categoriesRequest.getAll { [weak self] result in
  switch result {
  // 3
  case .failure:
    let message =
      "There was an error getting the categories"
    ErrorPresenter.showError(message: message, on: self)
  // 4
```

```
    case .success(let categories):
      self?.categories = categories
      DispatchQueue.main.async { [weak self] in
        self?.tableView.reloadData()
      }
    }
  }
```

这是它的作用：

1. 为类别创建ResourceRequest。
2. 从API获取所有类别。
3. 如果获取失败，则显示错误消息。
4. 如果获取成功，则填充categories数组并重新加载列表数据。

打开**AcronymRequest.swift**并在AcronymUserRequestResult下面添加新枚举，以表示缩略词加入类别的结果：

```
enum CategoryAddResult {
  case success
  case failure
}
```

这只是定义成功和失败情况。接下来，在AcronymRequest的底部添加以下方法：

```
func add(
  category: Category,
  completion: @escaping (CategoryAddResult) -> Void
) {
  // 1
  guard let categoryID = category.id else {
    completion(.failure)
    return
  }
  // 2
  let url = resource
    .appendingPathComponent("categories")
    .appendingPathComponent("\(categoryID)")
  // 3
  var urlRequest = URLRequest(url: url)
  urlRequest.httpMethod = "POST"
  // 4
  let dataTask = URLSession.shared
    .dataTask(with: urlRequest) { _, response, _ in
      // 5
      guard
        let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 201
        else {
          completion(.failure)
```

```
            return
        }
        // 6
        completion(.success)
    }
    dataTask.resume()
  }
```

这是它的作用：

1. 确保类别具有有效ID，否则使用失败情况调用完成处理程序。

2. 构建请求的URL。

3. 创建URLRequest并将HTTP方法设置为**POST**。

4. 从共享URLSession创建数据任务。

5. 确保响应是HTTP响应，响应状态为201 Created。否则，使用失败情况调用完成处理程序。

6. 使用成功情况调用完成处理程序。

打开**AddToCategoryTableViewController.swift**并在文件末尾添加以下扩展：

```
// MARK: - UITableViewDelegate
extension AddToCategoryTableViewController {
  override func tableView(
    _ tableView: UITableView,
    didSelectRowAt indexPath: IndexPath
  ) {
    // 1
    let category = categories[indexPath.row]
    // 2
    guard let acronymID = acronym.id else {
      let message = """
        There was an error adding the acronym
        to the category - the acronym has no ID
        """
      ErrorPresenter.showError(message: message, on: self)
      return
    }
    // 3
    let acronymRequest = AcronymRequest(acronymID: acronymID)
    acronymRequest
      .add(category: category) { [weak self] result in
        switch result {
        // 4
        case .success:
          DispatchQueue.main.async { [weak self] in
            self?.navigationController?
              .popViewController(animated: true)
```

```
            }
          // 5
          case .failure:
            let message = """
              There was an error adding the acronym
              to the category
              """
            ErrorPresenter.showError(message: message, on: self)
          }
      }
    }
  }
```

这是这个函数的作用：

1. 获得用户选择的类别。
2. 确保缩略词具有有效的ID；否则，显示错误消息。
3. 创建一个AcronymRequest以将缩略词加入到类别中。
4. 如果请求成功，则返回到上一个视图。
5. 如果请求失败，则显示错误消息。

最后，打开**AcronymDetailTableViewController.swift**以设置AddToCategoryTableViewController。在prepare(for:sender:)的末尾，添加一个新的segue identifier情况：

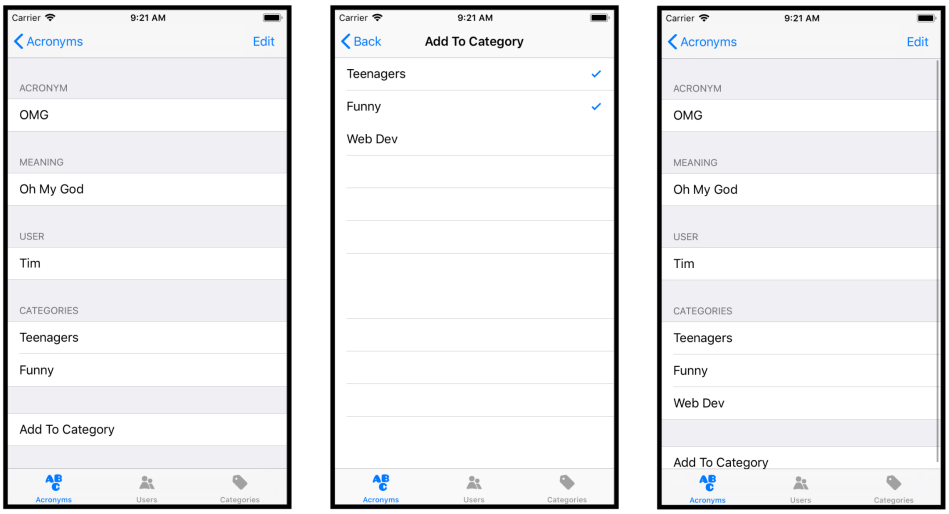
```
  else if segue.identifier == "AddToCategorySegue" {
    // 1
    guard let destination = segue.destination
      as? AddToCategoryTableViewController else {
        return
    }
    // 2
    destination.acronym = acronym
    destination.selectedCategories = categories
  }
```

这是它的作用：

1. 确保目标控制器是AddToCategoryTableViewController。
2. 在目标控制器上设置acronym和selectedCategories属性。

构建并运行。点击缩略词，然后在详细视图中，会出现标题为**Add To Category**的新行。点击此单元格，将显示类别列表，已选择的类别已被标记。

选择一个新类别，视图将关闭。现在缩略词详细信息视图在其类别列表中包含了新类别：



然后去哪儿?

本章向您展示了如何构建与Vapor API交互的iOS应用程序。但是，该应用程序功能不全，您可以改进它。例如，您可以添加类别信息视图，以显示特定类别的所有缩略词。

本书的下一节将向您展示如何构建另一个客户端：网站。