Chapter 22: Google Authentication

By Tim Condon

In the previous chapters, you learned how to add authentication to the TIL web site. However, sometimes users don't want to create extra accounts for an application and would prefer to use their existing accounts.

In this chapter, you'll learn how to use OAuth 2.0 to delegate authentication to Google, so users can log in with their Google accounts instead.

OAuth 2.0

OAuth 2.0 is an authorization framework that allows third-party applications to access resources on behalf of a user. Whenever you log in to a website with your Google account, you're using OAuth.

When you click **Login with Google**, Google is the site that authenticates you. You then authorize the application to have access to your Google data, such as your email. Once you've allowed the application access, Google gives the application a token. The app uses this token to authenticate requests to Google APIs. You'll implement this technique in this chapter.

Note: You must have a Google account to complete this chapter. If you don't have one, visit https://accounts.google.com/SignUp to create one.

Imperial

Writing all the necessary scaffolding to interact with Google's OAuth system and get a token is a time-consuming job!

There's a community package called Imperial, https://github.com/vapor-community/Imperial, that does the heavy lifting for you. It has integrations for Google, Facebook and GitHub with more planned.

Adding to your project

Open **Package.swift** in Xcode to add the new dependency. Replace `.package(url: "https://github.com/vapor/auth.git", from: "2.0.0")` with the following:

```
.package(url: "https://github.com/vapor/auth.git",
         from: "2.0.0"),
.package(url: "https://github.com/vapor-community/Imperial.git",
         from: "0.7.1")
```

Next, add the dependency to your `App` target's dependency array, after `"Authentication"`:

```
dependencies: ["FluentPostgreSQL",
               "Vapor",
               "Leaf",
               "Authentication",
               "Imperial"]
```

In Terminal, create a file for a new controller to manage Imperial's routes:

```
touch Sources/App/Controllers/ImperialController.swift
```

Finally, in Terminal, regenerate the Xcode project to pull in the new dependency and include the new file:

```
vapor xcode -y
```

When the project opens in Xcode, open **ImperialController.swift** and create a new empty controller:

```
import Vapor
import Imperial
import Authentication

struct ImperialController: RouteCollection {
  func boot(router: Router) throws {

  }
}
```

This creates a new type, `ImperialController`, that conforms to `RouteCollection`, implementing the required `boot(router:)`.

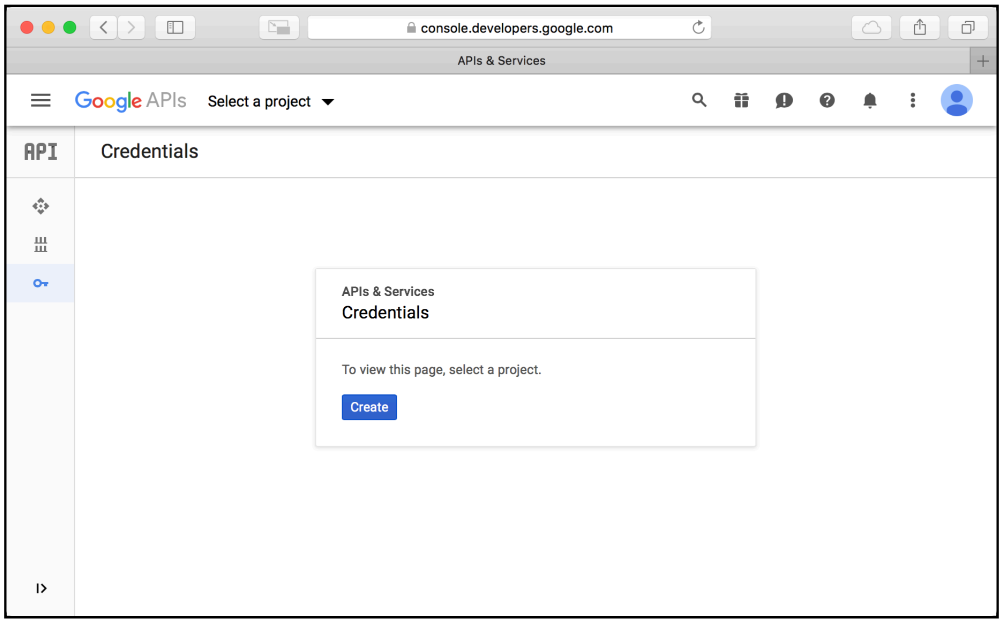
Open **routes.swift** and add the controller to your application at the bottom of `routes(_:)`:

```
let imperialController = ImperialController()
try router.register(collection: imperialController)
```

Setting up your application with Google

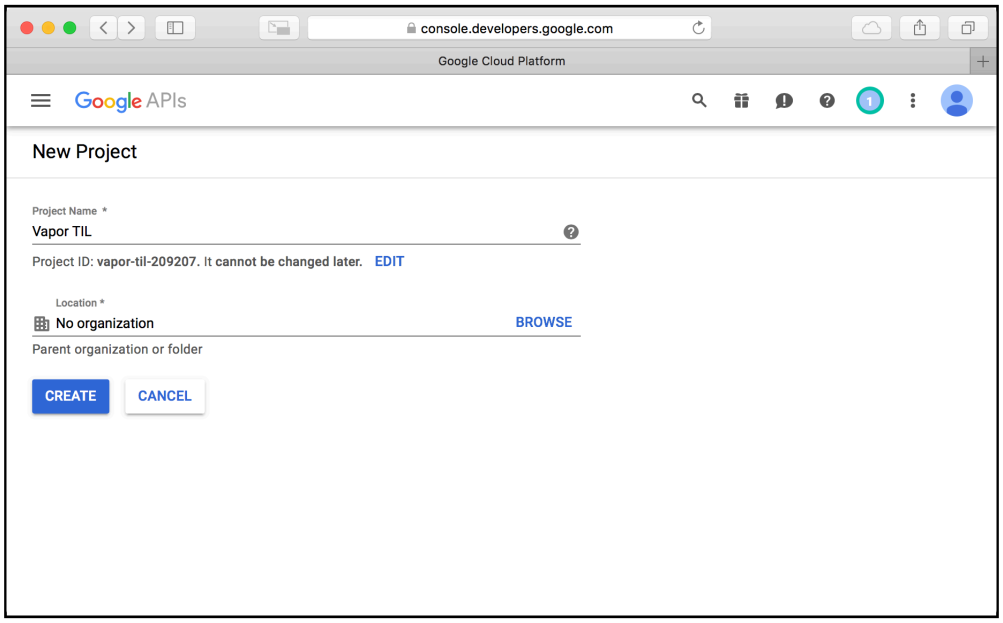
To be able to use Google OAuth in your application, you must first register the application with Google. In your browser, go to https://console.developers.google.com/apis/credentials.

If this is the first time you've used Google's credentials, the site prompts you to create a project:



To view this page, select a project.

Click **Create** to create a project for the TIL application. Fill in the form with an appropriate name, e.g. **Vapor TIL**:



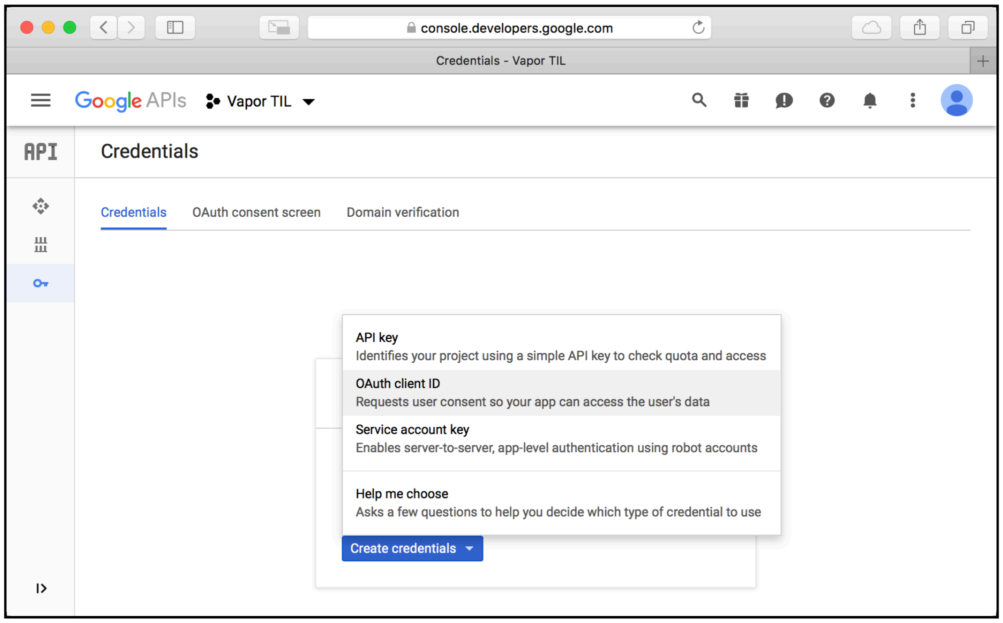
New Project

Project Name *
Vapor TIL

Project ID: vapor-til-209207. It cannot be changed later. EDIT

Location *
No organization BROWSE
Parent organization or folder

After it creates the project, the site takes you back to the Google credentials page. This time, click **Select** and select the created project. Click **Create Credentials** to create credentials for the TIL app and choose **OAuth client ID**:



API key
Identifies your project using a simple API key to check quota and access

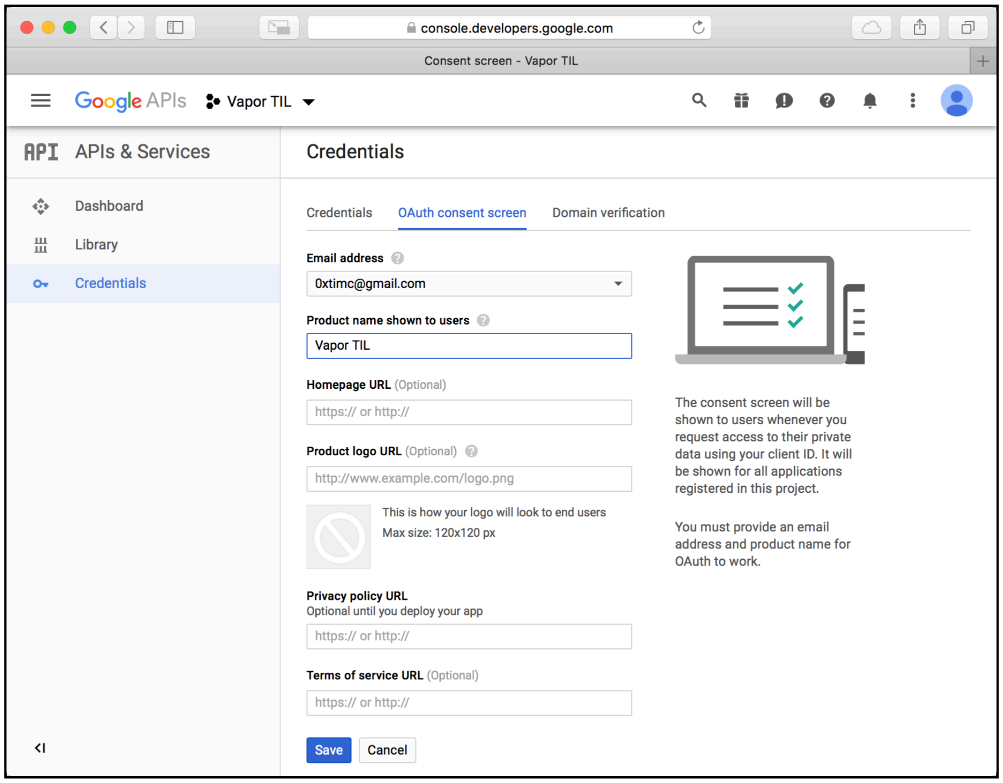
OAuth client ID
Requests user consent so your app can access the user's data

Service account key
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

Next click **Configure consent screen** to set up the page Google presents to users, so they can allow your application access to their details.

Add a product name and click **Save**:



Email address
0xtimc@gmail.com

Product name shown to users
Vapor TIL

Homepage URL (Optional)
https:// or http://

Product logo URL (Optional)
http://www.example.com/logo.png

This is how your logo will look to end users
Max size: 120x120 px

Privacy policy URL
Optional until you deploy your app
https:// or http://

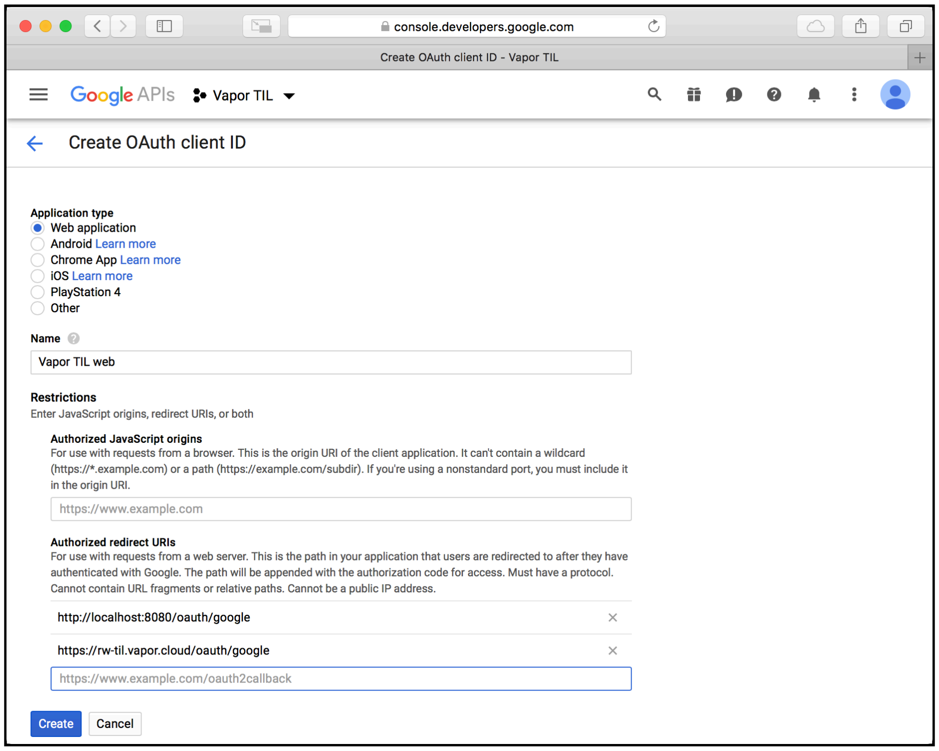
Terms of service URL (Optional)
https:// or http://

The consent screen will be shown to users whenever you request access to their private data using your client ID. It will be shown for all applications registered in this project.

You must provide an email address and product name for OAuth to work.

When creating a client ID, choose **Web application**. Add a redirect URI for your application for testing — **http://localhost:8080/oauth/google**. This is the URL that Google redirects back to once users have allowed your application access to their data.

If you want to deploy your application to the internet, such as with Vapor Cloud or Heroku, add another redirect for the URL for that site — e.g., **https://rw-til.vapor.cloud/oauth/google**:


console.developers.google.com
Create OAuth client ID - Vapor TIL
Google APIs
Vapor TIL

Create OAuth client ID

Application type
- [✓ Web application]
- Android Learn more
- Chrome App Learn more
- iOS Learn more
- PlayStation 4
- Other

Name: Vapor TIL web

Restrictions
Enter JavaScript origins, redirect URIs, or both

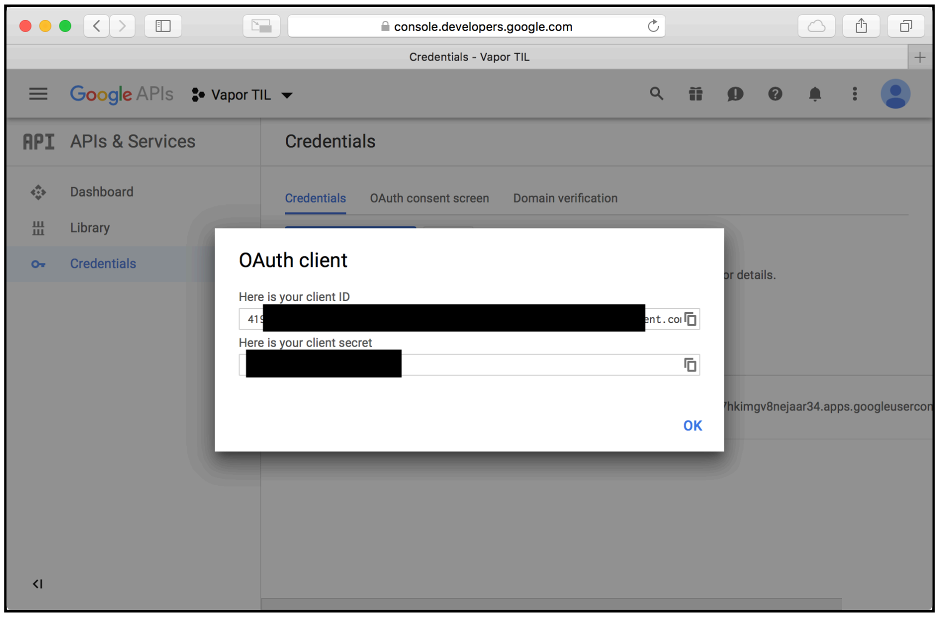
Authorized JavaScript origins
For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (https://*.example.com) or a path (https://example.com/subdir). If you're using a nonstandard port, you must include it in the origin URI.

https://www.example.com

Authorized redirect URIs
For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

http://localhost:8080/oauth/google
https://rw-til.vapor.cloud/oauth/google
https://www.example.com/oauth2callback

Click **Create** and the site gives you with your client ID and client secret:



OAuth client

Here is your client ID
41[redacted]ent.co

Here is your client secret
[redacted]

> **Note**: You *must* keep these safe and secure. Your secret allows you access to Google's APIs, and you should not share or check the secret into source control. You should treat it like a password.

Setting up the integration

Now that you've registered your application with Google, you can start integrating Imperial. Open **ImperialController.swift** and add the following under `boot(router:)`:

```
func processGoogleLogin(request: Request, token: String)
  throws -> Future<ResponseEncodable> {
    return request.future(request.redirect(to: "/"))
}
```

This defines a method to handle the Google login. The handler simply redirects the user to the home page — the same way that the regular login works. Imperial uses this method as the final callback once it has handled the Google redirect. Notice the use of `request.future(_:)` to create a future from `request.redirect(to:)`, since the function that Imperial uses requires a `Future`.

Next, set up the Imperial routes by adding the following in `boot(router:)`:

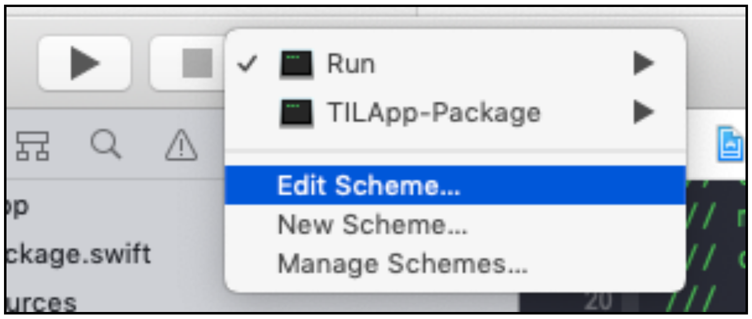
```
guard let googleCallbackURL =
  Environment.get("GOOGLE_CALLBACK_URL") else {
    fatalError("Google callback URL not set")
}
try router.oAuth(
  from: Google.self,
  authenticate: "login-google",
  callback: googleCallbackURL,
  scope: ["profile", "email"],
  completion: processGoogleLogin)
```

Here's what this does:

- Get the callback URL for Google from an environment variable — this is the URL you set up in the Google console.
- Register Imperial's Google OAuth router with your app's router.
- Tell Imperial to use the Google handlers.
- Set up the `/login-google` route as the route that triggers the OAuth flow. This is the route the application uses to allow users to log in via Google.
- Provide the callback URL to Imperial.

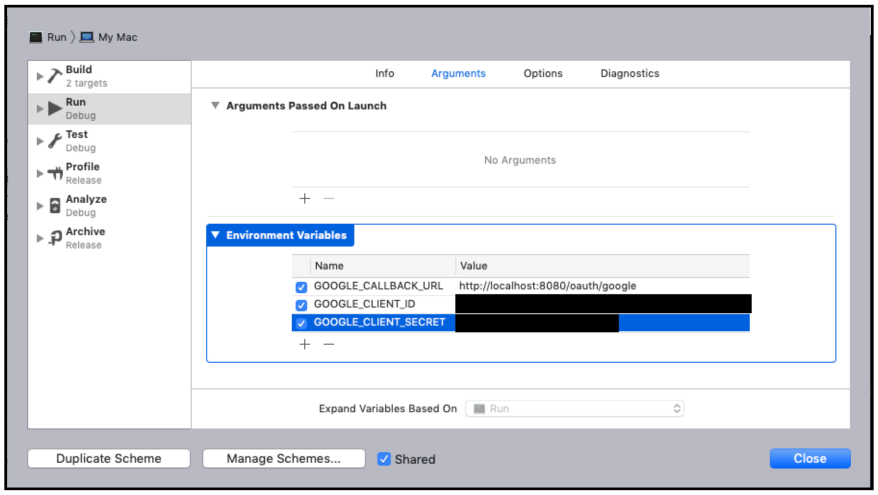
- Request the **profile** and **email** scopes from Google — the application needs these to create a user.
- Set the completion handler to `processGoogleLogin(request:token:)` - the method you created above.

In order for Imperial to work, you need to provide it the client ID and client secret that Google gave you. You provide these to Imperial using environment variables. To do this in Xcode, click the **Run** scheme, then click **Edit scheme**:



Under the **Arguments** tab, add three new **Environment Variables** as shown below:

- **GOOGLE_CALLBACK_URL**: **http://localhost:8080/oauth/google** — this is the URL you provided to Google.
- **GOOGLE_CLIENT_ID**: The client ID provided by Google.
- **GOOGLE_CLIENT_SECRET**: The client secret provided by Google.



Name	Value
✓ GOOGLE_CALLBACK_URL	http://localhost:8080/oauth/google
✓ GOOGLE_CLIENT_ID	[redacted]
✓ GOOGLE_CLIENT_SECRET	[redacted]

Integrating with web authentication

It's important to provide a seamless experience for users and match the experience for the regular login. To do this, you need to create a new user when a user logs in with Google for the first time. To create a user, you can use Google's API to get the necessary details using the OAuth token.

Sending requests to third-party APIs

At the bottom of **ImperialController.swift**, add a new type to decode the data from Google's API:

```
struct GoogleUserInfo: Content {
  let email: String
  let name: String
}
```

The request to Google's API returns many fields. However, you only care about the email, which becomes the username, and the name.

Next, under `GoogleUserInfo`, add the following:

```
extension Google {
  // 1
  static func getUser(on request: Request)
    throws -> Future<GoogleUserInfo> {
      // 2
      var headers = HTTPHeaders()
      headers.bearerAuthorization =
        try BearerAuthorization(token: request.accessToken())

      // 3
      let googleAPIURL =
        "https://www.googleapis.com/oauth2/v1/userinfo?alt=json"
      // 4
      return try request
        .client()
        .get(googleAPIURL, headers: headers)
        .map(to: GoogleUserInfo.self) { response in
        // 5
        guard response.http.status == .ok else {
          // 6
          if response.http.status == .unauthorized {
            throw Abort.redirect(to: "/login-google")
          } else {
            throw Abort(.internalServerError)
          }
        }
        // 7
        return try response.content
          .syncDecode(GoogleUserInfo.self)
      }
    }
  }
```

Here’s what this does:

1. Add a new function to Imperial’s `Google` service that gets a user’s details from the Google API.
2. Set the headers for the request by adding the OAuth token to the authorization header.
3. Set the URL for the request — this is Google’s API to get the user’s information.
4. Use `request.client()` to create a client to send a request. `get()` sends an HTTP GET request to the URL provided. Unwrap the returned future response.
5. Ensure the response status is **200 OK**.
6. Otherwise, return to the login page if the response was **401 Unauthorized** or return an error.
7. Decode the data from the response to `GoogleUserInfo` and return the result.

Next, replace the contents of `processGoogleLogin(request:token:)` with the following:

```
// 1
return try Google
  .getUser(on: request)
  .flatMap(to: ResponseEncodable.self) { userInfo in
  // 2
  return User
    .query(on: request)
    .filter(\.username == userInfo.email)
    .first()
    .flatMap(to: ResponseEncodable.self) { foundUser in

    guard let existingUser = foundUser else {
      // 3
      let user = User(name: userInfo.name,
                      username: userInfo.email,
                      password: UUID().uuidString)
      // 4
      return user
        .save(on: request)
        .map(to: ResponseEncodable.self) { user in
        // 5
        try request.authenticateSession(user)
        return request.redirect(to: "/")
      }
    }
    // 6
    try request.authenticateSession(existingUser)
    return request.future(request.redirect(to: "/"))
    }
  }
```

Here's what the new code does:

1. Get the user information from Google.
2. See if the user exists in the database by looking up the email as the username.
3. If the user doesn't exist, create a new `User` using the name and email from the user information from Google. Set the password to a UUID string, since you don't need it. This ensures that no one can login to this account via a normal password login.
4. Save the user and unwrap the returned future.
5. Call `request.authenticateSession(_:)` to save the created user in the session so the website allows access. Redirect back to the home page.
6. If the user already exists, authenticate the user in the session and redirect to the home page.

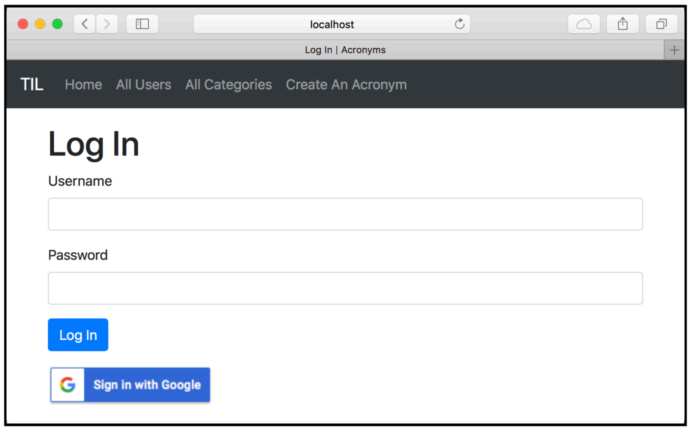
> Note: In a real world application, you may want to consider using a flag to separate out users registered on your site vs. logging in with social media.

The final thing to do is to add a button on the website to allow users to make use of the new functionality! Open **login.leaf** and, under `</form>`, add the following:

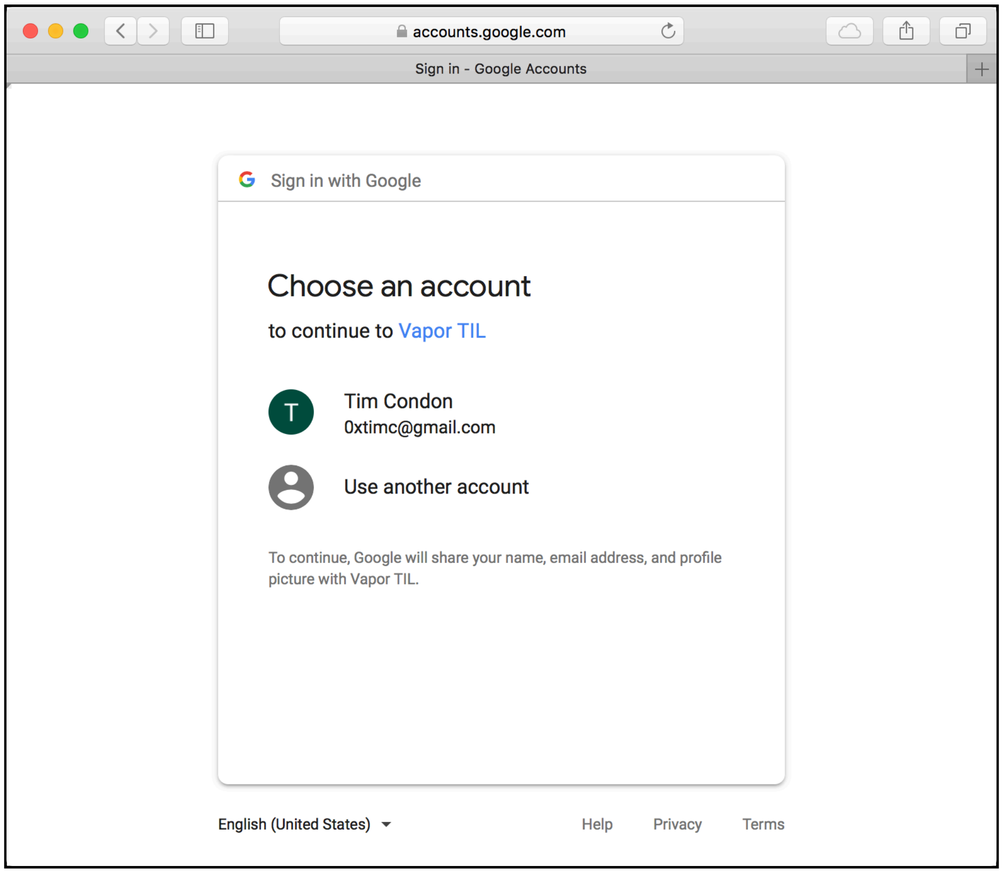
```
<a href="/login-google">
  <img class="mt-3" src="/images/sign-in-with-google.png"
   alt="Sign In With Google">
</a>
```

The sample project for this chapter contains a new, Google-provided image, **sign-in-with-google.png**, to display a **Sign in with Google** button. This adds the image as a link to **/login-google** — the route provided to Imperial to start the login. Save the Leaf template and build and run the application in Xcode. Visit **http://localhost:8080** in your browser.

Click **Create An Acronym** and the application takes you to the login page. You'll see the new **Sign in with Google** button:



Click the new button and the application takes you to a Google page to allow the TIL application access to your information:



To continue, Google will share your name, email address, and profile picture with Vapor TIL.

Select the account you want to use and the application redirects you back to the home page. Go to the **All Users** screen and you'll see your new user account. If you create an acronym, the application also uses that new user.

Where to go from here?

In this chapter, you learned how to integrate Google login into your website using Imperial and OAuth. This allows users to sign in with their existing Google accounts!

The next chapter shows you how to integrate another popular OAuth provider: GitHub.