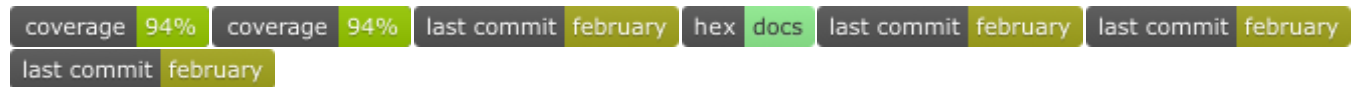

OAuth2 (Client)



An Elixir OAuth 2.0 Client Library.

Install

```
1 # mix.exs
2
3 defp deps do
4   # Add the dependency
5   [
6     {:oauth2, "~> 2.0"},
7     {:hackney, "~> 1.18"} # depending on what tesla adapter you use
8   ]
9 end
```

Configure a serializer

This library can be configured to handle encoding and decoding requests and responses automatically based on the `accept` and/or `content-type` headers.

If you need to handle various MIME types, you can simply register serializers like so:

```
1 OAuth2.Client.put_serializer(client, "application/vnd.api+json", Jason)
2 OAuth2.Client.put_serializer(client, "application/xml", MyApp.Parsers.XML)
```

The modules are expected to export `encode! /1` and `decode! /1`.

```
1 defmodule MyApp.Parsers.XML do
2   def encode!(data), do: # ...
3   def decode!(binary), do: # ...
4 end
```

Please see the documentation for `OAuth2.Serializer` for more details.

Configure a http client

The http client library used is `tesla`, the default adapter is `Httpc`, since it comes out of the box with every Erlang instance but you can easily change it to something better. You can configure another adaptor like this:

```
1 config :oauth2, adapter: Tesla.Adapter.Mint
```

You can also add your own tesla middleware:

```
1 config :oauth2, middleware: [  
2   Tesla.Middleware.Retry,  
3   {Tesla.Middleware.Fuse, name: :example}  
4 ]
```

Debug mode

Sometimes it's handy to see what's coming back from the response when getting a token. You can configure OAuth2 to output the response like so:

```
1 config :oauth2, debug: true
```

Usage

Current implemented strategies:

- Authorization Code
- Password
- Client Credentials

Authorization Code Flow (AuthCode Strategy)

```
1 # Initialize a client with client_id, client_secret, site, and  
  redirect_uri.  
2 # The strategy option is optional as it defaults to `OAuth2.Strategy.  
  AuthCode`.  
3  
4 client = OAuth2.Client.new([  
5   strategy: OAuth2.Strategy.AuthCode, #default  
6   client_id: "client_id",  
7   client_secret: "abc123",  
8   site: "https://auth.example.com",  
9   redirect_uri: "https://example.com/auth/callback"  
10 ])  
11  
12 # Generate the authorization URL and redirect the user to the provider.  
13 OAuth2.Client.authorize_url!(client)  
14 # => "https://auth.example.com/oauth/authorize?client_id=client_id&  
  redirect_uri=https%3A%2F%2Fexample.com%2Fauth%2Fcallback&  
  response_type=code"
```

```
15
16 # Use the authorization code returned from the provider to obtain an
    access token.
17 client = OAuth2.Client.get_token!(client, code: "someauthcode")
18
19 # Use the access token to make a request for resources
20 resource = OAuth2.Client.get!(client, "/api/resource").body
```

Client Credentials Flow

Getting an initial access token:

```
1 # Initializing a client with the strategy `OAuth2.Strategy.
    ClientCredentials`
2
3 client = OAuth2.Client.new([
4   strategy: OAuth2.Strategy.ClientCredentials,
5   client_id: "client_id",
6   client_secret: "abc123",
7   site: "https://auth.example.com"
8 ])
9
10 # Request a token from with the newly created client
11 # Token will be stored inside the `OAuth2.Client{}` struct (client.
    token)
12 client = OAuth2.Client.get_token!(client)
13
14 # client.token contains the `OAuth2.AccessToken{}` struct
15
16 # raw access token
17 access_token = client.token.access_token
```

Refreshing an access token:

```
1 # raw refresh token - use a client with `OAuth2.Strategy.Refresh` for
    refreshing the token
2 refresh_token = client.token.refresh_token
3
4 refresh_client = OAuth2.Client.new([
5   strategy: OAuth2.Strategy.Refresh,
6   client_id: "client_id",
7   client_secret: "abc123",
8   site: "https://auth.example.com",
9   params: %{"refresh_token" => refresh_token}
10 ])
11
12 # refresh_client.token contains the `OAuth2.AccessToken{}` struct
    again
13 refresh_client = OAuth2.Client.get_token!(refresh_client)
```

Write Your Own Strategy

Here's an example strategy for GitHub:

```
1 defmodule GitHub do
2   use OAuth2.Strategy
3
4   # Public API
5
6   def client do
7     OAuth2.Client.new([
8       strategy: __MODULE__,
9       client_id: System.get_env("GITHUB_CLIENT_ID"),
10      client_secret: System.get_env("GITHUB_CLIENT_SECRET"),
11      redirect_uri: "http://myapp.com/auth/callback",
12      site: "https://api.github.com",
13      authorize_url: "https://github.com/login/oauth/authorize",
14      token_url: "https://github.com/login/oauth/access_token"
15    ])
16    |> OAuth2.Client.put_serializer("application/json", Jason)
17  end
18
19  def authorize_url! do
20    OAuth2.Client.authorize_url!(client(), scope: "user,public_repo")
21  end
22
23  # you can pass options to the underlying http library via `opts`
  # parameter
24  def get_token!(params \\ [], headers \\ [], opts \\ []) do
25    OAuth2.Client.get_token!(client(), params, headers, opts)
26  end
27
28  # Strategy Callbacks
29
30  def authorize_url(client, params) do
31    OAuth2.Strategy.AuthCode.authorize_url(client, params)
32  end
33
34  def get_token(client, params, headers) do
35    client
36    |> put_header("accept", "application/json")
37    |> OAuth2.Strategy.AuthCode.get_token(params, headers)
38  end
39 end
```

Here's how you'd use the example GitHub strategy:

Generate the authorize URL and redirect the client for authorization.

```
1 GitHub.authorize_url!
```

Capture the `code` in your callback route on your server and use it to obtain an access token.

```
1 client = GitHub.get_token!(code: code)
```

Use the access token to access desired resources.

```
1 user = OAuth2.Client.get!(client, "/user").body
2
3 # Or
4 case OAuth2.Client.get(client, "/user") do
5   {:ok, %OAuth2.Response{body: user}} ->
6     user
7   {:error, %OAuth2.Response{status_code: 401, body: body}} ->
8     Logger.error("Unauthorized token")
9   {:error, %OAuth2.Error{reason: reason}} ->
10    Logger.error("Error: #{inspect reason}")
11 end
```

Examples

- Authenticate with Github (OAuth2/Phoenix)

License

The MIT License (MIT)

Copyright (c) 2015 Sonny Scroggin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.