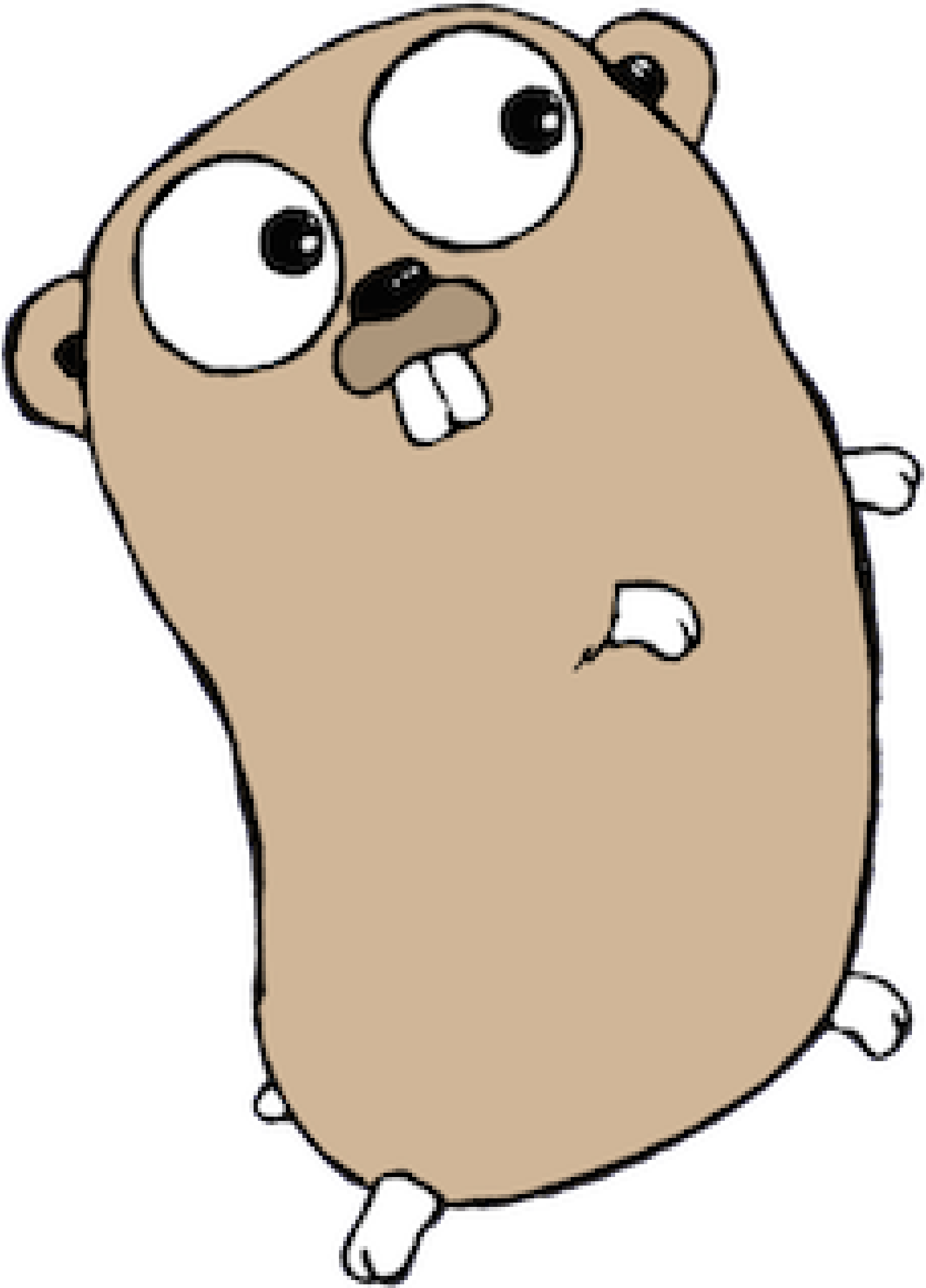


Accessing GitHub using Go

Satish Talim

This book is for sale at http://leanpub.com/accessinggithubusinggo

This version was published on 2015-04-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Also By Satish Talim

How Do I Write And Deploy Simple Web Apps With Go?

Building a package in Go

How do I use Sourcegraph with Go?

How do I use Sourcegraph with Ruby?

How to Deploy a Go Web App to the Google App Engine 101

How to Deploy a Go Web App to Heroku 101

How do I use the template package and handle forms?

Contents

Accessing GitHub using Go

GitHub, like many other sites, uses OAuth 2.0 protocol for authentication. OAuth2 is a protocol that lets external apps request authorization to private details in a user's GitHub account without getting their password. Applications that need to read or write private information using the API on behalf of another user should use OAuth.

GitHub API

First, read thro' the GitHub API documentation[1].

3-legged authorization

On a conceptual level it works in the following way:

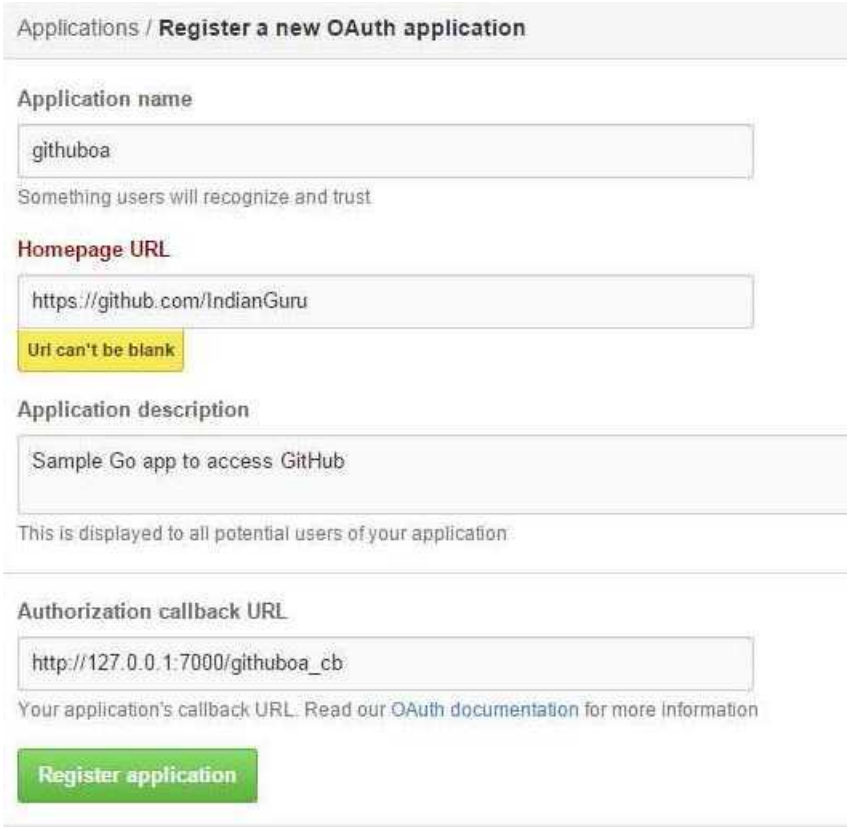
- Client has signed up to the GitHub server and got his client credentials (also known as consumer key and secret) ahead of time
- User wants to give the client access to his protected resources on the server
- Client retrieves the temporary credentials (also known as "request token") from the server
- Client redirects the resource owner to the server
- Resource owner grants the client access to his protected resources on the server
- Server redirects the user back to the client
- Client uses the temporary credentials to retrieve the token credentials (also known as "access token") from the server
- Client uses the token credentials to access the protected resources on the server

Register your app

Log into your GitHub a/c and register your app at https://github.com/settings/applications[2]. Click on the "Register new application" button. A registered OAuth application is assigned a unique Client ID and Client Secret. **The Client Secret should not be shared**. While registering, you can fill out every piece of information however you like, except the **Authorization callback URL**. This is easily the most important piece to setting up your application. It's the callback URL that GitHub returns the user to, after successful authentication.

I have registered my app `githuboa.go` at my GitHub a/c[3].

[1] https://developer.github.com/guides/getting-started/
[2] https://github.com/settings/applications
[3] https://github.com/IndianGuru



Register an app

Let's build our app githuboa.go

We shall be using oauth2[4] a Go package that contains a client implementation for the OAuth 2.0 specification.

It will be useful if you read the documentation for oauth2[5] and the OAuth[6] documentation of GitHub.

Now let's install `oauth2` and the `github` packages.

```
1 go get golang.org/x/oauth2
2 go get github.com/google/go-github/github
```

The flow of our app githuboa.go

- the user is on your website and clicks "Log into GitHub" link
- you redirect the user to GitHub's authorization page. In that url you specify desired access level and a random secret
- the user authorizes your app by clicking on a link

[4] https://github.com/golang/oauth2
[5] http://godoc.org/golang.org/x/oauth2
[6] https://developer.github.com/v3/oauth/

- GitHub redirects to a callback url on your website (which you provided when registering the app with GitHub)
- in the url handler, extract “secret” and “code” arguments
- you have to check that the secret is the same as the one you sent to GitHub (security measure that prevents forgery)
- you call another GitHub url to exchange code for access token

Access token is what you use to authenticate your API calls and allows you to make requests to the API on a behalf of a user.

Code: Create an OAuth config object

```
package main

import (
        "golang.org/x/oauth2"
        githuboauth "golang.org/x/oauth2/github"
)

var (
        // You must register the app at https://github.com/settings/applications
        // Set callback to http://127.0.0.1:7000/githuboa_cb
        // Set ClientId and ClientSecret to the values you got
        // after registering your app
        oauthConf = &oauth2.Config{
                ClientID:     "", // please enter your value
                ClientSecret: "", // please enter your value
                // Comma separated list of scopes
                // select level of access you want https://developer.github.com/v3/oauth/#scopes
                Scopes:   []string{"user:email"},
                Endpoint: githuboauth.Endpoint,
        }
        // An unguessable random string. It is used to protect against
        // cross-site request forgery attacks
        oauthStateString = "arandomstring"
)

func main() {
}
```

Config[7] used above describes a typical 3-legged OAuth2 flow, with both the client application information and the server's endpoint URLs. The Endpoint used above is defined here[8].

[7] http://godoc.org/golang.org/x/oauth2#Config
[8] https://github.com/golang/oauth2/blob/master/github/github.go

Your main html page

```
1  package main
2
3  import (
4          "fmt"
5          "net/http"
6
7          "golang.org/x/oauth2"
8          githuboauth "golang.org/x/oauth2/github"
9  )
10
11 const htmlIndex = `<html><body><p>Well, hello there!</p>
12 <p>We're going to now talk to the GitHub API. Ready?</p>
13 <p>Log into <a href="/login">GitHub</a></p>
14 </body></html>
15 `
16
17 var (
18         // You must register the app at https://github.com/settings/applications
19         // Set callback to http://127.0.0.1:7000/githuboa_cb
20         // Set ClientId and ClientSecret to the values you got
21         // after registering your app
22         oauthConf = &oauth2.Config{
23                 ClientID:     "", // please enter your value
24                 ClientSecret: "", // please enter your value
25                 // Comma separated list of scopes
26                 // select level of access you want https://developer.github.com/v3/oauth/#scopes
27                 Scopes:   []string{"user:email"},
28                 Endpoint: githuboauth.Endpoint,
29         }
30         // An unguessable random string. It is used to protect against
31         // cross-site request forgery attacks
32         oauthStateString = "arandomstring"
33 )
34
35 func main() {
36         http.HandleFunc("/", handleMain)
37
38         fmt.Print("Started running on http://127.0.0.1:7000\n")
39         fmt.Println(http.ListenAndServe(":7000", nil))
40 }
41
42 func handleMain(w http.ResponseWriter, r *http.Request) {
43         w.Header().Set("Content-Type", "text/html; charset=utf-8")
```

```
44          w.WriteHeader(http.StatusOK)
45          w.Write([]byte(htmlIndex))
46  }
```

Refer to the ResponseWriter interface[9] where the functions `Header()`, `WriteHeader()` and `Write()` are mentioned. Check the details of the Set[10] method.

[9] http://golang.org/pkg/net/http/#ResponseWriter
[10] http://golang.org/pkg/net/http/#Header

Login to GitHub

Once the user clicks on the "Log into GitHub" link the handler for /login url, redirects to GitHub's authorization page. GitHub will show the authorization page to your user. If the user authorizes your app, GitHub will re-direct to OAuth callback. Here's how you can turn it into a token, token into http client and use that client to list GitHub information about the user.

```
package main

import (
        "fmt"
        "net/http"

        "golang.org/x/oauth2"
        githuboauth "golang.org/x/oauth2/github"
        "html/template"
)

const htmlIndex = `<html><body><p>Well, hello there!</p>
<p>We're going to now talk to the GitHub API. Ready?</p>
<p>Log into <a href="/login">GitHub</a></p>
</body></html>
`

var userInfoTemplate = template.Must(template.New("").Parse(`
<html><body>
<p>This app is now authenticated to access your GitHub user info.</p>
<p>User details are:</p><p>
{{.}}
</p>
<p>That's it!</p>
</body></html>
`))

var (
        // You must register the app at https://github.com/settings/applications
        // Set callback to http://127.0.0.1:7000/githuboa_cb
        // Set ClientId and ClientSecret to the values you got
        // after registering your app
        oauthConf = &oauth2.Config{
                ClientID:     "", // please enter your value
                ClientSecret: "", // please enter your value
                // Comma separated list of scopes
                // select level of access you want https://developer.github.com/v3/oauth/#scopes
                Scopes:   []string{"user:email"},
```

```
39                  Endpoint: githuboauth.Endpoint,
40          }
41          // An unguessable random string. It is used to protect against
42          // cross-site request forgery attacks
43          oauthStateString = "arandomstring"
44  )
45
46  func main() {
47          http.HandleFunc("/", handleMain)
48          http.HandleFunc("/login", handleGitHubLogin)
49          http.HandleFunc("/githuboa_cb", handleGitHubCallback)
50
51          fmt.Print("Started running on http://127.0.0.1:7000\n")
52          fmt.Println(http.ListenAndServe(":7000", nil))
53  }
54
55  func handleMain(w http.ResponseWriter, r *http.Request) {
56          w.Header().Set("Content-Type", "text/html; charset=utf-8")
57          w.WriteHeader(http.StatusOK)
58          w.Write([]byte(htmlIndex))
59  }
60
61  // /login
62  func handleGitHubLogin(w http.ResponseWriter, r *http.Request) {
63          url := oauthConf.AuthCodeURL(oauthStateString, oauth2.AccessTypeOnline)
64          http.Redirect(w, r, url, http.StatusTemporaryRedirect)
65  }
66
67  // githuboa_cb. Called by github after authorization is granted
68  func handleGitHubCallback(w http.ResponseWriter, r *http.Request) {
69          // If the user accepts your request, GitHub redirects back
70          // to your site with a temporary code in a code parameter
71          // as well as the state you provided in the previous step
72          // in a state parameter. If the states don't match, the
73          // request has been created by a third party and the process
74          // should be aborted.
75          state := r.FormValue("state")
76          if state != oauthStateString {
77                  fmt.Printf("invalid oauth state, expected '%s', got '%s'\n", oauthStateString, state)
78                  http.Redirect(w, r, "/", http.StatusTemporaryRedirect)
79                  return
80          }
81
82          code := r.FormValue("code")
83
```

```
84          // On success, exchange this for an access token
85          token, err := oauthConf.Exchange(oauth2.NoContext, code)
86          if err != nil {
87                  fmt.Printf("oauthConf.Exchange() failed with '%s'\n", err)
88                  http.Redirect(w, r, "/", http.StatusTemporaryRedirect)
89                  return
90          }
91
92          oauthClient := oauthConf.Client(oauth2.NoContext, token)
93
94          // https://godoc.org/github.com/google/go-github/github
95          client := github.NewClient(oauthClient)
96
97          user, _, err := client.Users.Get("")
98          if err != nil {
99                  fmt.Printf("client.Users.Get() failed with '%s'\n", err)
100                 http.Redirect(w, r, "/", http.StatusTemporaryRedirect)
101                 return
102         }
103
104         buf := []string{"GitHub login id: ", *user.Login, "| GitHub email id: ", *user.Email}
105
106         userInfoTemplate.Execute(w, buf)
107 }
```

Read the details of AuthCodeURL[11]. `AuthCodeURL` returns a `URL` to OAuth 2.0 provider's consent page that asks for permissions for the required scopes explicitly. State is a token to protect the user from CSRF attacks. You must always provide a non-zero string. Opts[12] may include `AccessTypeOnline` or `AccessTypeOffline`, as well as `ApprovalForce`.

The Exchange[13] method above converts an authorization code into a token. It is used after a resource provider redirects the user back to the Redirect URI (the `URL` obtained from `AuthCodeURL`). The HTTP client to use is derived from the context. If a client is not provided via the context, `http.DefaultClient` is used. The code will be in the `*http.Request.FormValue("code")`. Before calling `Exchange`, be sure to validate `FormValue("state")`.

The Client[14] method above returns an HTTP client using the provided token. The token will auto-refresh as necessary. The underlying HTTP transport will be obtained using the provided context. The returned client and its Transport should not be modified.

`client := github.NewClient(oauthClient)` constructs a new GitHub client, then uses the various services on the client to access different parts of the GitHub API.

[11] http://godoc.org/golang.org/x/oauth2#Config.AuthCodeURL
[12] http://godoc.org/golang.org/x/oauth2#AuthCodeOption
[13] http://godoc.org/golang.org/x/oauth2#Config.Exchange
[14] http://godoc.org/golang.org/x/oauth2#Config.Client

`user, _, err := client.Users.Get("")` - see details here[15] fetches a user. Passing the empty string will fetch the authenticated user.

You can get all the details about a User[16].

Run the above program on http://127.0.0.1:7000[17].

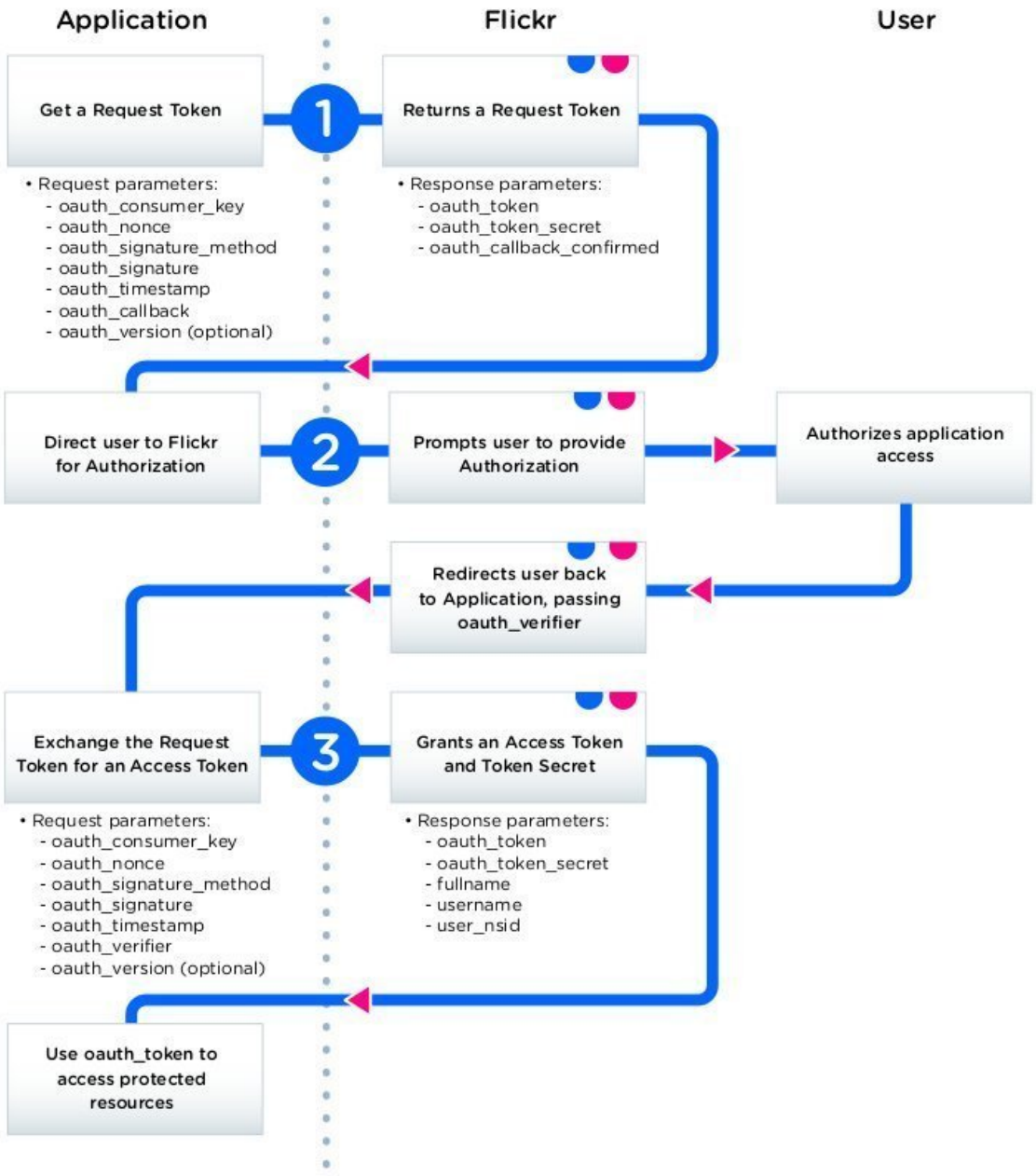
That's it!

You can download the entire program from here[18].

Exercise

How about writing an app that accesses Flickr using OAuth?

[15] https://godoc.org/github.com/google/go-github/github#UsersService.Get
[16] https://godoc.org/github.com/google/go-github/github#User
[17] http://127.0.0.1:7000
[18] https://gist.github.com/IndianGuru/2f686c755b4ec4cf67bb#file-githuboq-go



Flickr OAuth