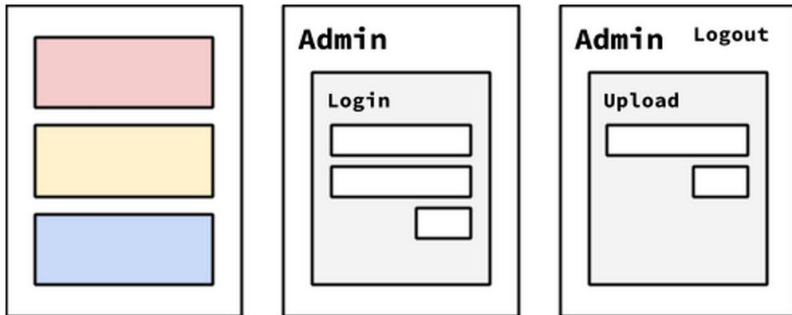



# Photo Blog

# Photo Blog


- Our goal for this project is to create a basic photo blog. It should have a few pages:
- The initial page (<https://localhost:8080/>) should list the most recent uploaded photos.
- Clicking a photo should download it. (hint: HTML5 download attribute or the Content-Disposition header)
- There should be an admin page (<https://localhost:8080/admin>) which allows a user to login (you can hardcode the username and password).
  - Once logged in the admin user should be presented with a form to upload an image.
  - The uploaded image should then be accessible on the main page.
  - The admin section should require TLS.
    - A self-signed certificate is ok. It's ok to require TLS for the regular section of the site as well.




one version


▼  12\_photo-blog


▼  01i


▼  assets


▼  imgs

 01.jpg

 02.jpg

 03.jpg

▼  templates

 index.gohtml

 login.gohtml

 upload-file.gohtml

 cert.pem

 key.pem

 main.go

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title></title>
6  </head>
7  <body>
8      <h1><a href="https://localhost:8080/login">LOG IN</a></h1>
9      <h1><a href="https://localhost:8080/upload">UPLOAD (only when logged in)</a></h1>
10     <h1><a href="https://localhost:8080/logout">LOG OUT</a></h1>
11         {{range .Pictures}}
12         |     
13         |     {{else}}
14         |     <h1>NO PICTURES</h1>
15         |     {{end}}
16     </body>
17 </html>
```

html body

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title></title>
6  </head>
7  <body>
8      <h1>LOG IN</h1>
9      <form method="post" action="https://localhost:8080/login">
10         <h3>User name</h3>
11         <input type="text" name="userName" id="userName">
12         <h3>Password</h3>
13         <input type="text" name="password" id="password">
14         <br>
15         <input type="submit">
16         <input type="submit" name="logout" value="logout">
17     </form>
18 </body>
19 </html>
20
21 <!--
22 from
23 49_cookies-sessions/08_log-in-out
24 -->
```

html body form

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title></title>
6  </head>
7  <body>
8      <form method="POST" enctype="multipart/form-data">
9          <input type="file" name="my-file">
10         <input type="submit">
11     </form>
12 </body>
13 </html>
14
15 <!--
16 from
17 48_passing-data/08_form_file-upload/04
18 -->
```

```
13
14 var err error
15 var tpl *template.Template
16 var store = sessions.NewCookieStore([]byte("something-very-secret"))
17
18 type Model struct {
19     Pictures []string
20 }
21
22 // get []string of picture paths
23 var Data Model = getPicturePaths()
24
```



```
113
114 func getPicturePaths() Model {
115     files := []string{}
116     // OR: files := make([]string, 0)
117     filepath.Walk("./", func(path string, fi os.FileInfo, err error) error {
118         // skip directories
119         if fi.IsDir() {
120             return nil
121         }
122         // for windows: replace \ with /
123         path = strings.Replace(path, "\\ ", "/", -1)
124         // I only want .jpg files
125         if strings.HasSuffix(path, ".jpg") {
126             files = append(files, path)
127         }
128         return nil
129     })
130     return Model{Pictures: files}
131 }
```

```
24
25 func main() {
26     // Parse templates
27     tpl, err = tpl.ParseGlob("assets/templates/*.gohtml")
28     if err != nil {
29         log.Fatalln(err)
30     }
31
32     http.HandleFunc("/", index)
33     http.HandleFunc("/login", login)
34     http.HandleFunc("/upload", upload)
35     http.HandleFunc("/logout", logout)
36     http.Handle("/assets/", http.StripPrefix("/assets", http.FileServer(http.Dir("./assets"))))
37     http.Handle("/favicon.ico", http.NotFoundHandler())
38
39
40     // to generate cert and key:
41     // go run $(go env GOROOT)/src/crypto/tls/generate_cert.go --host=localhost
42     http.ListenAndServeTLS(":8080", "cert.pem", "key.pem", nil)
43 }
```

```
44  
45 func index(res http.ResponseWriter, req *http.Request) {  
46     tpl.ExecuteTemplate(res, "index.gohtml", Data)  
47 }  
48
```

```
2
3 import (
4     "net/http"
5     "github.com/gorilla/sessions"
6     "html/template"
7     "log"
8     "os"
9     "io"
10    "path/filepath"
11    "strings"
12 )
13
```

```
48
49 func login(res http.ResponseWriter, req *http.Request) {
50     session, _ := store.Get(req, "session")
51
52     // PROCESS FORM SUBMISSION
53     if req.Method == "POST" {
54         password := req.FormValue("password")
55         if password == "secret" {
56             session.Values["logged_in"] = true
57         } else {
58             http.Error(res, "invalid credentials", 401)
59             return
60         }
61         // save session
62         session.Save(req, res)
63         // redirect to main page
64         http.Redirect(res, req, "/", 302)
65     }
66
67     // Execute template
68     tpl.ExecuteTemplate(res, "login.gohtml", nil)
69 }
```

```
70
71 func logout(res http.ResponseWriter, req *http.Request) {
72     session, _ := store.Get(req, "session")
73     delete(session.Values, "logged_in")
74     session.Save(req, res)
75     http.Redirect(res, req, "/", 302)
76 }
77
```



```

78 func upload(res http.ResponseWriter, req *http.Request) {
79     // redirect if not logged in
80     session, _ := store.Get(req, "session")
81     if session.Values["logged_in"] == false ||
82         session.Values["logged_in"] == nil {
83         http.Redirect(res, req, "https://localhost:8080/login", 302)
84     }
85
86     if req.Method == "POST" {
87         // <input type="file" name="my-file">
88         src, hdr, err := req.FormFile("my-file")
89         if err != nil {
90             http.Error(res, err.Error(), 500)
91             return
92         }
93         defer src.Close()
94
95         // create a new file
96         path := "/Users/tm002/Documents/go/src/github.com/goestoelever
/imgs/"
97         dst, err := os.Create(path + hdr.Filename)
98         if err != nil {
99             http.Error(res, err.Error(), 500)
100             return
101         }
102         defer dst.Close()
103
104         // copy the uploaded file into the new file
105         io.Copy(dst, src)
106         Data = getPicturePaths()
107         http.Redirect(res, req, "/", 302)
108     }
109
110     // Execute template
111     tpl.ExecuteTemplate(res, "upload-file.gohtml", nil)
112 }
113

```

another version




▼  03i

▼  assets

▼  imgs

 CANADA 023.jpg

 IMG\_20150703\_165948.jpg

 notes.txt

 spring\_09 078.jpg


 to\_upload.jpg

▼  tpl


 admin\_login.gohtml

 admin\_upload.gohtml

 index.gohtml

 cert.pem

 changes.txt

 key.pem

 main.go

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Pictures</title>
6  </head>
7  <body>
8      <h1>PICTURES</h1>
9
10     {{ if .LoggedIn }}
11     <a href="/admin/logout">Log Out</a>
12     {{ .LoggedIn }}
13     {{ else }}
14     {{ .LoggedIn }}
15     <p>We are in the else</p>
16     {{ end }}
17
18     <a href="/admin">admin</a>
19     {{ range .Photos }}
20     <br/>
21     <a href="#" download="{{ .PhotoPath }}"></a>
22
23     <!--Google Static Maps Developer Guide-->
24     <!--https://developers.google.com/maps/documentation/static-maps/intro-->
25     
28     {{ end }}
29 </body>
30 </html>
```

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8      <form method="post">
9          <h3>User name</h3>
10         <input type="text" name="userName" id="userName">
11         <h3>Password</h3>
12         <input type="text" name="password" id="password">
13         <br>
14         <input type="submit">
15     </form>
16 </body>
17 </html>
```

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8      {{ if . }}
9      <a href="/admin/logout">Log Out</a>
10     {{ . }}
11     {{ else }}
12     {{ . }}
13     <p>We are in the else</p>
14     {{ end }}
15
16
17  <form method="POST" enctype="multipart/form-data">
18      <label for="file">Choose File To Upload</label>
19      <input type="file" id="file" name="file">
20      <br>
21      <input type="submit">
22
23  </form>
24  </body>
25  </html>
```

```
16
17 var tpl *template.Template
18 var store = sessions.NewCookieStore([]byte("something-very-secret"))
19
20
21 func init() {
22     var err error
23     tpl, err = template.ParseFiles("assets/tpl/index.gohtml", "assets/tpl/admin_login.gohtml", "assets/tpl/admin_upload.gohtml")
24     if err != nil {
25         log.Fatalln("couldn't parse", err)
26     }
27 }
28
```

```
28
29 func main() {
30     http.HandleFunc("/", home)
31     http.HandleFunc("/admin", admin)
32     http.HandleFunc("/admin/upload", upload)
33     http.HandleFunc("/admin/logout", logout)
34     http.Handle("/assets/imgs/", http.StripPrefix("/assets/imgs/", http.FileServer(http.Dir("assets/imgs/"))))
35     http.ListenAndServe(":9000", context.ClearHandler(http.DefaultServeMux))
36     /*
37     go run $(go env GOROOT)/src/crypto/tls/generate_cert.go --host=somedomainname.com
38     */
39 }
40
```

```
40
41 func home(res http.ResponseWriter, req *http.Request) {
42
43     type Photo struct {
44         PhotoPath string
45         Lat float64
46         Long float64
47     }
48
49     var model struct {
50         Photos []Photo
51         LoggedIn bool
52     }
53
54     // session
55     session, _ := store.Get(req, "session-name")
56     _, model.LoggedIn = session.Values["loggedin"]
57
```



```

41 func home(res http.ResponseWriter, req *http.Request) {
42
43     type Photo struct { ... }
44
45     var model struct { ... }
46
47     // session
48     session, _ := store.Get(req, "session-name")
49     _, model.LoggedIn = session.Values["loggedin"]
50
51     filepath.Walk("assets/imgs", func(path string, info os.FileInfo, err error) error {
52         if !info.IsDir() {
53
54             fmt.Println("WALKING", path)
55
56             var currentPhoto Photo
57
58             currentPhoto.PhotoPath = path
59
60             // photo lat long
61             fname := "assets/imgs/us.jpg"
62
63             f, err := os.Open(path)
64             if err != nil {
65                 log.Fatal(err)
66             }
67
68             // Optionally register camera makenote data parsing - currently Nikon and
69             // Canon are supported.
70             exif.RegisterParsers(mknote.All...)
71
72             x, err := exif.Decode(f)
73             if err != nil {
74                 log.Println("no info")
75                 return nil
76             }
77         }
78     })
79 }

```



```
2
3 import (
4     "github.com/gorilla/context"
5     "github.com/gorilla/sessions"
6     "html/template"
7     "io"
8     "log"
9     "net/http"
10    "os"
11    "path/filepath"
12    "github.com/rwcarlsen/goexif/exif"
13    "github.com/rwcarlsen/goexif/mknote"
14    "fmt"
15 )
16
```

```

84 //      camModel, _ := x.Get(exif.Model) // normally, don't ignore errors!
85 //      fmt.Println(camModel.StringVal())
86 //
87 //      focal, _ := x.Get(exif.FocalLength)
88 //      number, denom, _ := focal.Rat2(0) // retrieve first (only) rat. value
89 //      fmt.Printf("%v/%v", number, denom)
90 //
91 //      // Two convenience functions exist for date/time taken and GPS coords:
92 //      tm, _ := x.DateTime()
93 //      fmt.Println("Taken: ", tm)
94 //
95
96 currentPhoto.Lat , currentPhoto.Long, _ = x.LatLong()
97 fmt.Println("lat, long: ", currentPhoto.Lat , " ", " ", currentPhoto.Long)
98
99 model.Photos = append(model.Photos, currentPhoto)
100 }
101 return nil
102 })
103
104 err := tpl.ExecuteTemplate(res, "index.gohtml", model)
105 if err != nil {
106     http.Error(res, err.Error(), 500)
107 }
108 }
109

```

```

type Photo struct {
    PhotoPath string
    Lat float64
    Long float64
}

```

```

var model struct {
    Photos []Photo
    LoggedIn bool
}

```

index.gohtml

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Pictures</title>
6 </head>
7 <body>
8     <h1>PICTURES</h1>
9
10     {{ if .LoggedIn }}
11     <a href="/admin/logout">Log Out</a>
12     {{ .LoggedIn }}
13     {{ else }}
14     {{ .LoggedIn }}
15     <p>We are in the else</p>
16     {{ end }}
17
18     <a href="/admin">admin</a>
19     {{ range .Photos }}
20     <br/>
21     <a href="#" download="{{ .PhotoPath }}"></a>
22
23     <!--Google Static Maps Developer Guide-->
24     <!--https://developers.google.com/maps/documentation/static-maps/intro-->
25     
28     {{ end }}
29 </body>
30 </html>

```

```

109 func admin(res http.ResponseWriter, req *http.Request) {
110     userName := req.FormValue("userName")
111     password := req.FormValue("password")
112
113     if userName == "You" && password == "Me" {
114         // Get a session. We're ignoring the error resulted from decoding an
115         // existing session: Get() always returns a session, even if empty.
116         session, _ := store.Get(req, "session-name")
117         // Set some session values.
118         session.Values["loggedin"] = "true"
119         // Save it.
120         session.Save(req, res)
121         http.Redirect(res, req, "/admin/upload", 302)
122         return
123     }
124
125     tpl.ExecuteTemplate(res, "admin_login.gohtml", nil)
126 }
127
128

```

admin\_login.gohtml x

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8     <form method="post">
9         <h3>User name</h3>
10        <input type="text" name="userName" id="userName">
11        <h3>Password</h3>
12        <input type="text" name="password" id="password">
13        <br>
14        <input type="submit">
15    </form>
16 </body>
17 </html>

```

```

128 func upload(res http.ResponseWriter, req *http.Request) {
129     // Get a session. We're ignoring the error resulted from decoding an
130     // existing session: Get() always returns a session, even if empty.
131     session, _ := store.Get(req, "session-name")
132     // Set some session values.
133     _, ok := session.Values["loggedin"]
134     if !ok {
135         //Change url
136         http.Redirect(res, req, "/admin", 302)
137         return
138     }
139
140     // if they are uploading a file, handle that
141     if req.Method == "POST" {
142         src, hdr, err := req.FormFile("file")
143         if err != nil {
144             panic(err)
145         }
146         defer src.Close()
147
148         fileName := hdr.FileName
149         dst, err := os.Create("assets/imgs/" + fileName)
150         if err != nil {
151             http.Error(res, err.Error(), 500)
152             return
153         }
154         defer dst.Close()
155
156         io.Copy(dst, src)
157     }
158
159     // execute template
160     tpl.ExecuteTemplate(res, "admin_upload.gohtml", ok)
161 }

```

```

admin_upload.gohtml x
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8     {{ if . }}
9     <a href="/admin/logout">Log Out</a>
10    {{ . }}
11    {{ else }}
12    {{ . }}
13    <p>We are in the else</p>
14    {{ end }}
15
16
17 <form method="POST" enctype="multipart/form-data">
18     <label for="file">Choose File To Upload</label>
19     <input type="file" id="file" name="file">
20     <br>
21     <input type="submit">
22
23 </form>
24 </body>
25 </html>

```

```
163
164 func logout(res http.ResponseWriter, req *http.Request) {
165     // Get a session. We're ignoring the error resulted from decoding an
166     // existing session: Get() always returns a session, even if empty.
167     session, _ := store.Get(req, "session-name")
168     delete(session.Values, "loggedin")
169     session.Save(req, res)
170     http.Redirect(res, req, "/admin", 302)
171     return
172 }
173
```

“This is not a small deal that you’re able to make this. Very few people in the world are able to build a web app like this. It requires a lot of specialized knowledge.”

~ Caleb Doxsey