

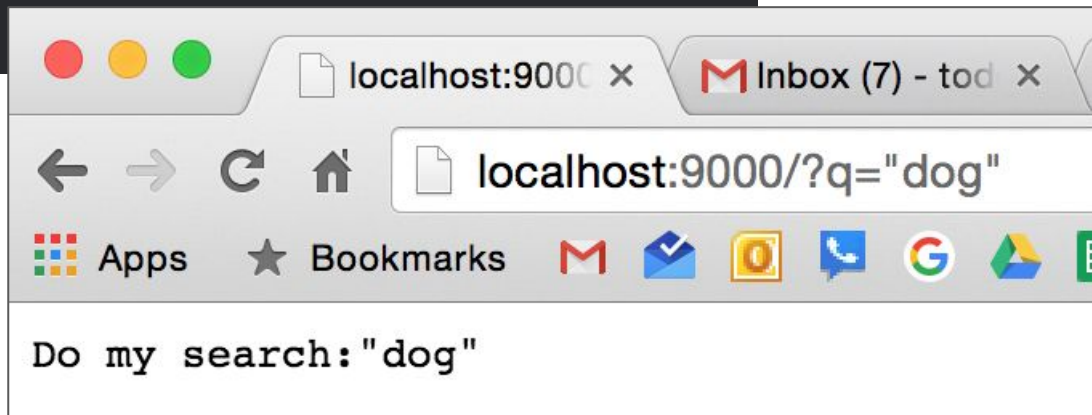
Passing Data

passing data to the HTTP server via
URL & Form Submission

URL Query String & Form Request Body

URL Query String









```
main.go x
1 package main
2
3 import (
4     "net/http"
5     "io"
6 )
7
8 func main() {
9     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
10         key := "q"
11         val := req.URL.Query().Get(key)
12         io.WriteString(res, "Do my search:" + val)
13     })
14     http.ListenAndServe(":9000", nil)
15 }
16
17 // visit this page:
18 // http://localhost:9000/?q="dog"
19
```



















Query String Downside

- limited amounts of data can be passed through the query string
- easily accessible to the user
 - the user can modify
- messy URLs

Request Body
(form submission)

godoc.org/net/http#Request.FormValue

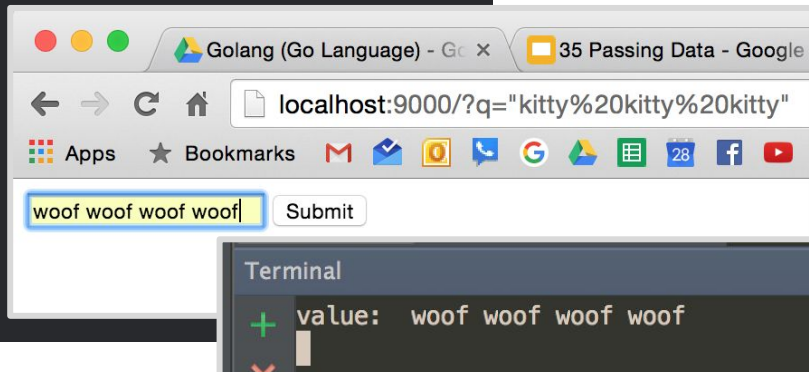
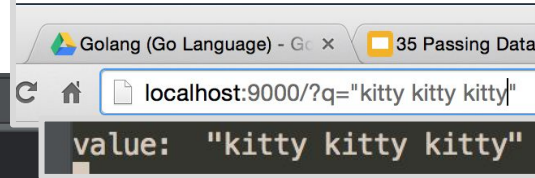
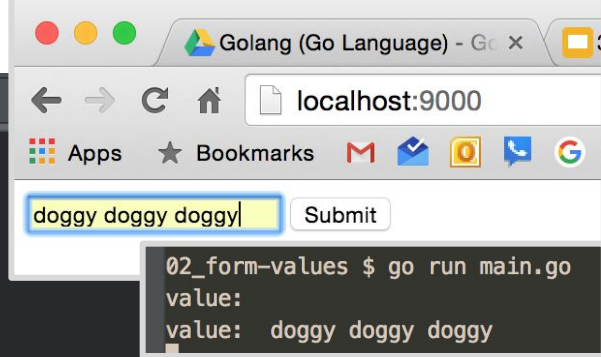
BookmarksPMHawk»C

func (*Request) FormValue

```
func (r *Request) FormValue(key string) string
```

FormValue returns the first value for the named component of the query. POST and PUT body parameters take precedence over URL query string values. FormValue calls ParseMultipartForm and ParseForm if necessary and ignores any errors returned by these functions. If key is not present, FormValue returns the empty string. To access multiple values of the same key, call ParseForm and then inspect Request.Form directly.

```
1 package main
2
3 import (
4     "net/http"
5     "io"
6     "fmt"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         key := "q"
12         val := req.FormValue(key)
13         fmt.Println("value: ", val)
14         res.Header().Set("Content-Type", "text/html")
15         io.WriteString(res, `<form method="POST">
16
17         <input type="text" name="q">
18         <input type="submit">
19
20         </form>`)
21     })
22     http.ListenAndServe(":9000", nil)
23 }
24 }
```



🔍

📱

Elements

Network

Sources

Timeline

Profiles

Resources

Audits

Console

NetBeans

>

⚙️

🖼️

✕

 View:
☐ Preserve log ☒ Disable cache | No throttling

Name

localhost

Headers	Preview	Response	Timing
---------	---------	----------	--------

▼General

Remote Address: [::1]:9000

Request URL: http://localhost:9000/

Request Method: POST

Status Code: ● 200 OK

[Response Headers](#) [view source](#)

Content-Length: 89

Content-Type: text/html
Date: Fri, 13 Jun 2015 00:36:43 GMT

Date: Tue, 29 Sep 2015 09:26:42 GMT

Request headers	view source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp;q=0.8	

Accept-encoding: gzip, deflate

Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

Cache-Control: no-cache

Connection: keep-alive

Content-Length: 19

Content-Type: application/x-www-form-urlencoded

Host: localhost:9000

Origin: http://localhost:9000

```

#pragma: no-cache

```

```
Referer: http://localhost:9000/
```

Upgrade-Insecure-Requests: 1

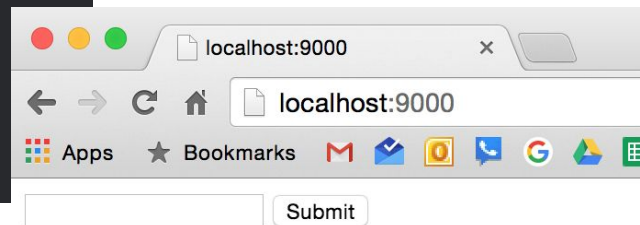
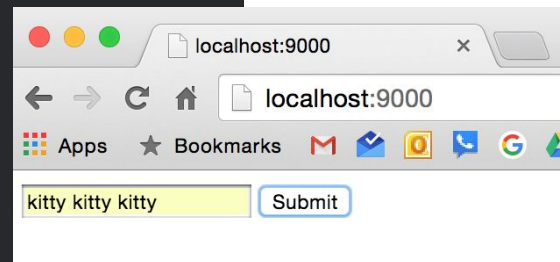
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36

Form Data view source view URL encoded

q. সুগ্গ সুগ্গ সুগ্গ

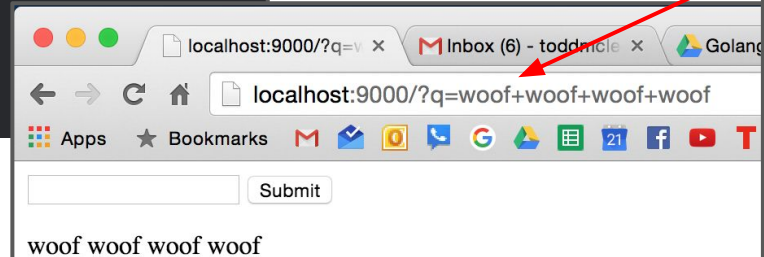
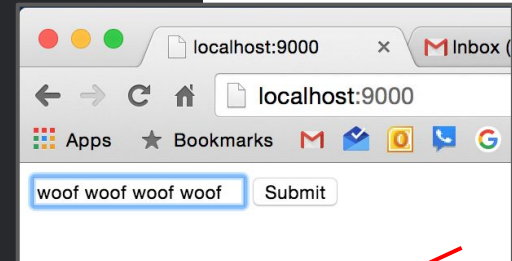
```
main.go x
1 package main
2
3 import (
4     "net/http"
5     "io"
6     "fmt"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         key := "q"
12         val := req.FormValue(key)
13         fmt.Println("value: ", val)
14         res.Header().Set("Content-Type", "text/html")
15         io.WriteString(res, `<form method="POST">
16
17             <input type="text" name="q">
18             <input type="submit">
19
20         </form>` + val)
21     })
22     http.ListenAndServe(":9000", nil)
23 }
24
```

POST vs ...

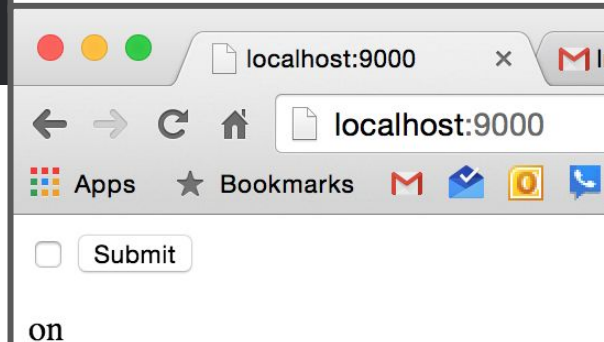
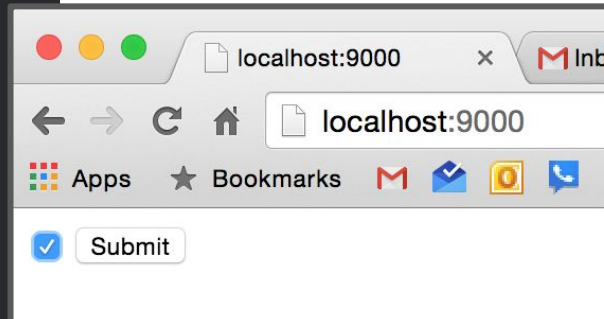


kitty kitty kitty

```
1 package main
2
3 import (
4     "net/http"           POST vs ... GET
5     "io"                 FORM ... URL
6     "fmt"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         key := "q"
12         val := req.FormValue(key)
13         fmt.Println("value: ", val)
14         res.Header().Set("Content-Type", "text/html")
15         io.WriteString(res, `<form method="GET">
16
17             <input type="text" name="q">
18             <input type="submit">
19
20         </form>` + val)
21     })
22     http.ListenAndServe(":9000", nil)
23 }
```



```
main.go x
1 package main
2
3 import (
4     "net/http"
5     "io"
6     "fmt"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         key := "q"
12         val := req.FormValue(key)
13         fmt.Println("value: ", val)
14         res.Header().Set("Content-Type", "text/html")
15         io.WriteString(res, `<form method="POST">
16
17         <input type="checkbox" name="q">
18         <input type="submit">
19
20         </form>` + val)
21     })
22     http.ListenAndServe(":9000", nil)
23 }
```



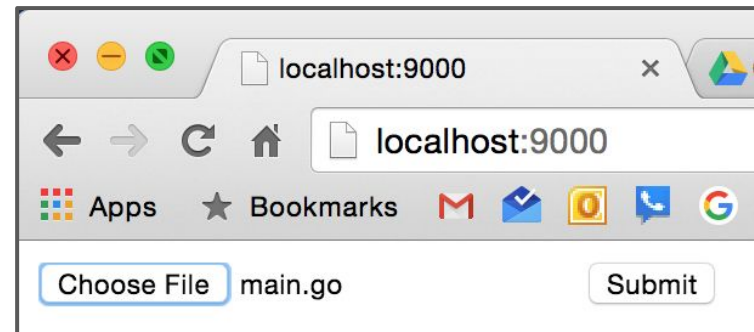
File Upload

(form submission)

```

1 package main
2
3 import (
4     "net/http"
5     "io"
6     "fmt"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         key := "q"
12         file, hdr, err := req.FormFile(key)
13         fmt.Println(file, hdr, err)
14         res.Header().Set("Content-Type", "text/html")
15
16         // you have to put this enctype for uploading files
17         io.WriteString(res, `<form method="POST" enctype="multipart/form-data">
18             <input type="file" name="q">
19             <input type="submit">
20         </form>`)
21     })
22     http.ListenAndServe(":9000", nil)
23 }

```



Terminal

```

+ 02_form-values $ cd ../05_form-values/
05_form-values $ go run main.go
<nil> <nil> request Content-Type isn't multipart/form-data
{0xc820017110} &{main.go map[Content-Disposition:[form-data; name="q"; filename="main.go"]] Content-Type:[application/octet-stream]] [112 97 99 107 97 103 101 32 109 97 105 110 10 10 105 109 112 11
1 114 116 32 40 10 9 34 110 101 116 47 104 116 116 112 34 10 9 34 105 111 34 10 9 34 102 109 116 34 10 41 10 10 102 117 110 99 32 109 97 105 110 40 41 32 123 10 9 104 116 116 112 46 72 97 110 100
108 101 70 117 110 99 40 34 47 34 44 32 102 117 110 99 40 114 101 115 32 104 116 116 112 46 82 101 115 112 111 110 115 101 87 114 105 116 101 114 44 32 114 101 113 32 42 104 116 116 112 46 82 101
113 117 101 115 116 41 32 123 10 9 9 107 101 121 32 58 61 32 34 113 34 10 9 9 102 105 108 101 44 32 104 100 114 44 32 101 114 114 32 58 61 32 114 101 113 46 70 111 114 109 70 105 108 101 40 107 10
1 121 41 10 9 9 102 109 116 46 80 114 105 110 116 108 110 40 102 105 108 101 44 32 104 100 114 44 32 101 114 114 41 10 9 9 114 101 115 46 72 101 97 100 101 114 40 41 46 83 101 116 40 34 67 111 110
116 101 110 116 45 84 121 112 101 34 44 32 34 116 101 120 116 47 104 116 109 108 34 41 10 10 9 9 47 47 32 121 111 117 32 104 97 118 101 32 116 111 32 112 117 116 32 116 104 105 115 32 101 110 99
116 121 112 101 32 102 111 114 32 117 112 108 111 97 100 105 110 103 32 102 105 108 101 115 10 9 9 105 111 46 87 114 105 116 101 83 116 114 105 110 103 40 114 101 115 44 32 96 60 102 111 114 109 3
2 109 101 116 104 111 100 61 34 80 79 83 84 34 32 101 110 99 116 121 112 101 61 34 109 117 108 116 105 112 97 114 116 47 102 111 114 109 45 100 97 116 97 34 62 10 32 32 32 32 32 60 105 110 112
117 116 32 116 121 112 101 61 34 102 105 108 101 34 32 110 97 109 101 61 34 113 34 62 10 32 32 32 32 32 60 105 110 112 117 116 32 116 121 112 101 61 34 115 117 98 109 105 116 34 62 10 32 32 32
32 60 47 102 111 114 109 62 96 41 10 9 125 41 10 9 104 116 116 112 46 76 105 115 116 101 110 65 110 100 83 101 114 118 101 40 34 58 57 48 48 48 34 44 32 110 105 108 41 10 125] } <nil>

```

https://godoc.org/net/http

func (*Request) FormFile

```
func (r *Request) FormFile(key string) (multipart.File, *multipart.FileHeader, error)
```

FormFile returns the first file for the provided form key. FormFile calls ParseMultipartForm and ParseForm if necessary.

https://godoc.org/mime/multipart#File

type File

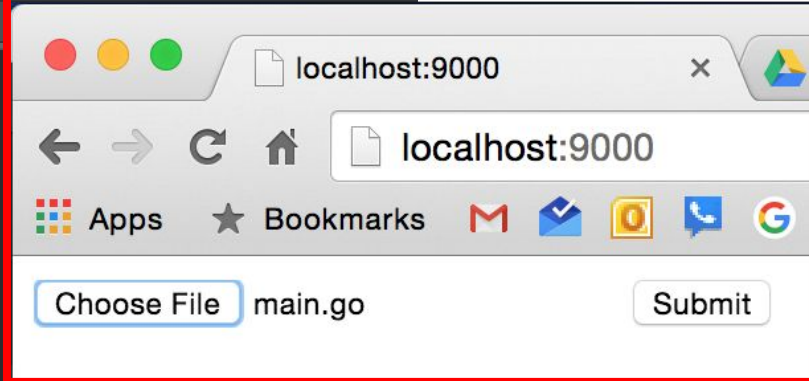
```
type File interface {  
    io.Reader  
    io.ReaderAt  
    io.Seeker  
    io.Closer  
}
```

Basically a “read only” file

File is an interface to access the file part of a multipart message. Its contents may be either stored in memory or on disk. If stored on disk, the File's underlying concrete type will be an *os.File.


```
1 package main
2
3 import (
4     "fmt"
5     "io"
6     "io/ioutil"
7     "net/http"
8 )
9
10 func main() {
11     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
12         key := "q"
13         file, _, err := req.FormFile(key)
14         if err != nil {
15             panic(err)
16         }
17         defer file.Close()
18
19         bs, _ := ioutil.ReadAll(file)
20
21         fmt.Println(string(bs))
22         res.Header().Set("Content-Type", "text/html")
23         // you have to put this enctype for uploading files
24         io.WriteString(res, `
```

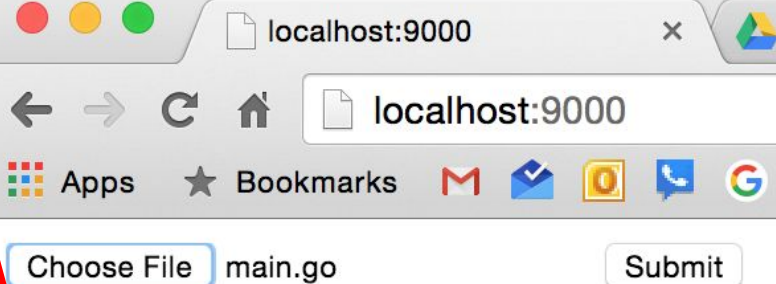
```
1 package main
2
3 import (
4     "fmt"
5     "io"
6     "io/ioutil"
7     "net/http"
8 )
9
10 func main() {
11     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
12         key := "q"
13         file, _, err := req.FormFile(key)
14         if err != nil {
15             panic(err)
16         }
17         defer file.Close()
18
19         bs, _ := ioutil.ReadAll(file)
20
21         fmt.Println(string(bs))
22         res.Header().Set("Content-Type", "text/html")
23         // you have to put this enctype for uploading files
24         io.WriteString(res, `<form method="POST" enctype="multipart/form-data">
25 <input type="file" name="q">
26 <input type="submit">
27 </form>`)
28     })
29     http.ListenAndServe(":9000", nil)
30 }
```




```
1 package main
```

```
2
3 import (
4     "fmt"
5     "io"
6     "io/ioutil"
7     "net/http"
```

```
8 )
9
10 func main() {
11     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
12         key := "q"
13         file, err := req.FormFile(key)
14         if err != nil {
15             panic(err)
16         }
17         defer file.Close()
18         bs, _ := ioutil.ReadAll(file)
19         fmt.Println(string(bs))
20         res.Header().Set("Content-Type", "text/html")
21         // you have to put this enctype for uploading files
22         io.WriteString(res, `<form method="POST" enctype="multipart/form-data">
23             <input type="file" name="q">
24             <input type="submit">
25         </form>`)
26         http.ListenAndServe(":9000", nil)
27     })
28 }
29
30 }
```



```
<http.Request) {
```

```
<input type="file" name="q">
<input type="submit">
</form>`
})
http.ListenAndServe(":9000", nil)
```

```
1 package main
2
3 import (
4     "io"
5     "net/http"
6     "os"
7 )
8
9 func main() {
10     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11         if req.Method == "POST" {
12             key := "q"
13             file, _, err := req.FormFile(key)
14             if err != nil {
15                 panic(err)
16             }
17             defer file.Close()
18             io.Copy(os.Stdout, file)
19         }
20
21         res.Header().Set("Content-Type", "text/html")
22         // you have to put this enctype for uploading files
23         io.WriteString(res, `<form method="POST" enctype="multipart/form-data">
24             <input type="file" name="q">
25             <input type="submit">
26         </form>`)
27     })
28     http.ListenAndServe(":9000", nil)
29 }
30
31
```

Review

- `Request.URL.Query().Get(KEY)`
- `Request.FormValue(KEY)`
- `Request.FormFile(KEY)`

exercises

Form Submission

Create an http application which has a page with a form that takes in first and last name and on submit displays “Hello <FIRST> <LAST>”

```

11 type Person struct {
12     FirstName string
13     LastName  string
14 }
15
16 func main() {
17
18     // parse template
19     tpl, err := template.ParseFiles("form.gohtml")
20     if err != nil {
21         log.Fatalln(err)
22     }
23
24     http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
25         // receive form submission
26         fName := req.FormValue("first")
27         lName := req.FormValue("last")
28         fmt.Println("fName: ", fName)
29         fmt.Println("[[]byte(fName): ", []byte(fName))
30         fmt.Println("typeOf: ", reflect.TypeOf(fName))
31
32         // execute template
33         err = tpl.Execute(res, Person{fName, lName})
34         if err != nil {
35             http.Error(res, err.Error(), 500)
36             log.Fatalln(err)
37         }
38     })
39
40     http.ListenAndServe(":9000", nil)
41 }

```

```

main.go x form.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
9 <form method="POST">
10     <label for="firstName">First Name</label>
11     <input type="text" id="firstName" name="first">
12     <br>
13     <label for="lastName">Last Name</label>
14     <input type="text" id="lastName" name="last">
15     <br>
16     <input type="submit">
17 </form>
18
19 <h1>{{_.FirstName }} {{_.LastName }}</h1>
20 </body>
21 </html>

```

Document

localhost:9000

Apps Bookmarks

First Name

Last Name

Terminal

```
+ fName:
[]byte(fName): []
✗ typeOf: string
fName: James
[]byte(fName): [74 97 109 101 115]
typeOf: string
```

Document

localhost:9000

Apps Bookmarks

First Name

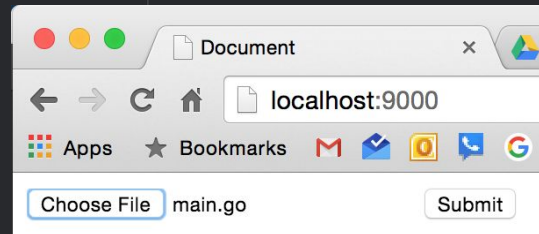
Last Name

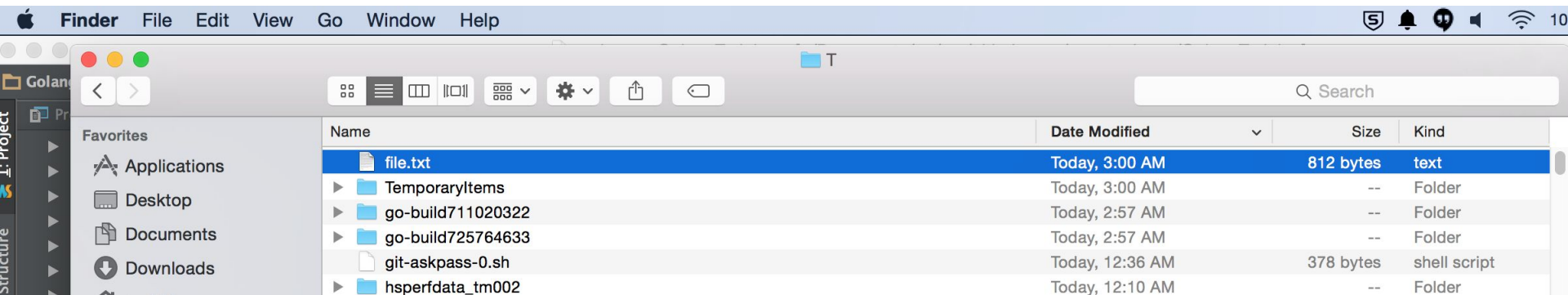
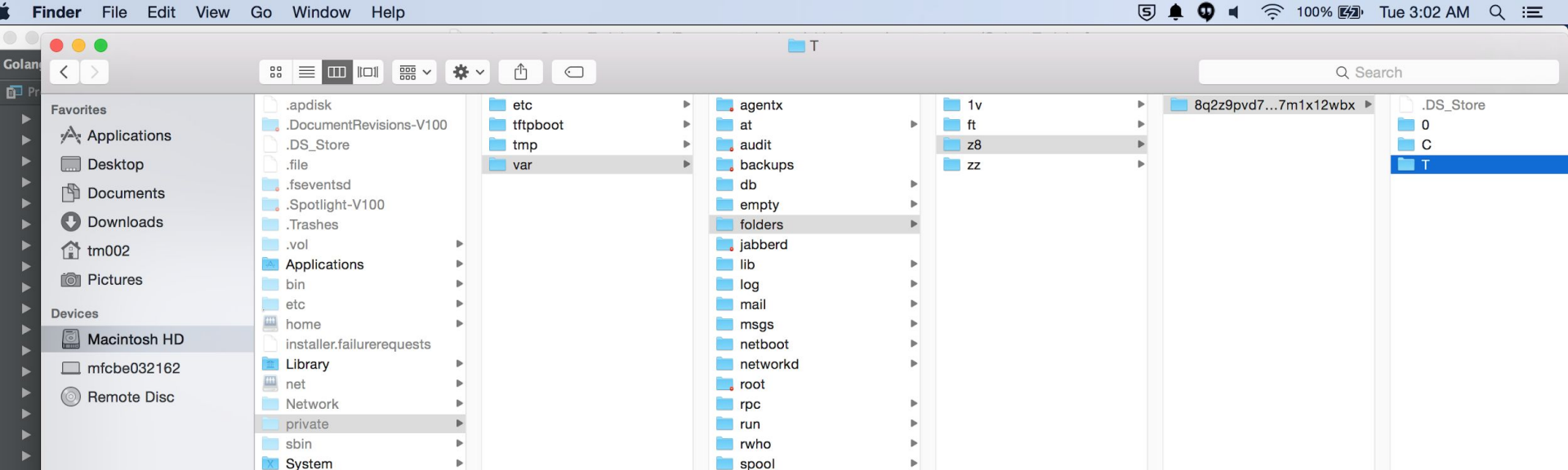
James Bond

File Upload

Create an http application with a form that accepts a file and uploads it to “/tmp”.


```
10 }
11 func main() {
12     fmt.Println("TEMP DIR:", os.TempDir())
13     http.ListenAndServe(":9000", http.HandlerFunc(func(res http.ResponseWriter, req *http.Request) {
14         if req.Method == "POST" {
15             src, _, err := req.FormFile("my-file")
16             if err != nil {
17                 http.Error(res, err.Error(), 500)
18                 return
19             }
20             defer src.Close()
21
22             dst, err := os.Create(filepath.Join(os.TempDir(), "file.txt"))
23             if err != nil {
24                 http.Error(res, err.Error(), 500)
25                 return
26             }
27             defer dst.Close()
28
29             io.Copy(dst, src)
30         }
31
32         res.Header().Set("Content-Type", "text/html")
33         io.WriteString(res, `
34 <form method="POST" enctype="multipart/form-data">
35     <input type="file" name="my-file">
36     <input type="submit">
37 </form>
38 `)
39     })
40 }
```





```
package main

import (
    "net/http"
    "io"
    "os"
    "path/filepath"
    "fmt"
)

func main() {
    fmt.Println("TEMP DIR:", os.TempDir())
    http.ListenAndServe(":9000", http.HandlerFunc(func(res http.ResponseWriter, req
*http.Request) {
        if req.Method == "POST" {
            src, _, err := req.FormFile("my-file")
            if err != nil {
                http.Error(res, err.Error(), 500)
                return
            }
            defer src.Close()

            dst, err := os.Create(filepath.Join(os.TempDir(), "file.txt"))
            if err != nil {
                http.Error(res, err.Error(), 500)
                return
            }
            defer dst.Close()

            io.Copy(dst, src)
        }
    })
}
```

Another solution:

```
1 func main() {
2
3     // parse template
4     tpl, err := template.ParseFiles("form.gohtml")
5     if err != nil {
6         log.Fatalln(err)
7     }
8
9     // handler
10    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
11        // receive form submission
12        if req.Method == "POST" {
13            src, _, err := req.FormFile("file")
14            if err != nil {
15                panic(err)
16            }
17            defer src.Close()
18
19            dst, err := os.Create(filepath.Join(".", "file.txt"))
20            if err != nil {
21                http.Error(res, err.Error(), 500)
22                return
23            }
24            defer dst.Close()
25
26            io.Copy(dst, src)
27
28            // execute template
29            err = tpl.Execute(res, nil)
30            if err != nil {
31                http.Error(res, err.Error(), 500)
32                log.Fatalln(err)
33            }
34        }
35    })
36
37    http.ListenAndServe(":9000", nil)
38 }
```

io.LimitReader

```
1 package main
2
3
4 import (
5     "net/http"
6     "io"
7     "os"
8     "path/filepath"
9 )
10 func main() {
11     http.ListenAndServe(":9000", http.HandlerFunc(func(res http.ResponseWriter, req *http.Request) {
12         if req.Method == "POST" {
13             file, _, err := req.FormFile("my-file")
14             if err != nil {
15                 http.Error(res, err.Error(), 500)
16                 return
17             }
18             defer file.Close()
19
20             src := io.LimitReader(file, 400)
21
22             dst, err := os.Create(filepath.Join(".", "file.txt"))
23             if err != nil {
24                 http.Error(res, err.Error(), 500)
25                 return
26             }
27             defer dst.Close()
28
29             io.Copy(dst, src)
30
31             res.Header().Set("Content-Type", "text/html")
32             io.WriteString(res, `
33 <form method="POST" enctype="multipart/form-data">
34   <input type="file" name="my-file">
35   <input type="submit">
36 </form>
37 `)
38         })
39     })
40 }
```