Handling HTTP Requests

Alex Schultz
SOFTWARE ENGINEER | AWS ML HERO
@AlexCSchultz

Overview

Creating Basic Handlers

Working with JSON

Request

URL Path

Middleware

CORS

Update the Client Application

Handling HTTP Requests

http.Handle

Registers a handler to handle requests matching a patten

http.HandleFunc

Registers a function to handle requests matching a pattern

http.Handle

```
func Handle(pattern string, handler Handler)


type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

main.go

```
import “net/http”


type fooHandler struct {
    Message string
}
func (f *fooHandler) ServeHTTP(w http.ResponseWriter, r *http.Request){
  w.Write([]byte(f.Message))
}
func main() {
  http.Handle(“/foo”, &fooHandler{Message: “hello world”})
}
```

http.HandleFunc

```
func HandleFunc(pattern string, handler func(ResponseWriter, *Request))
```

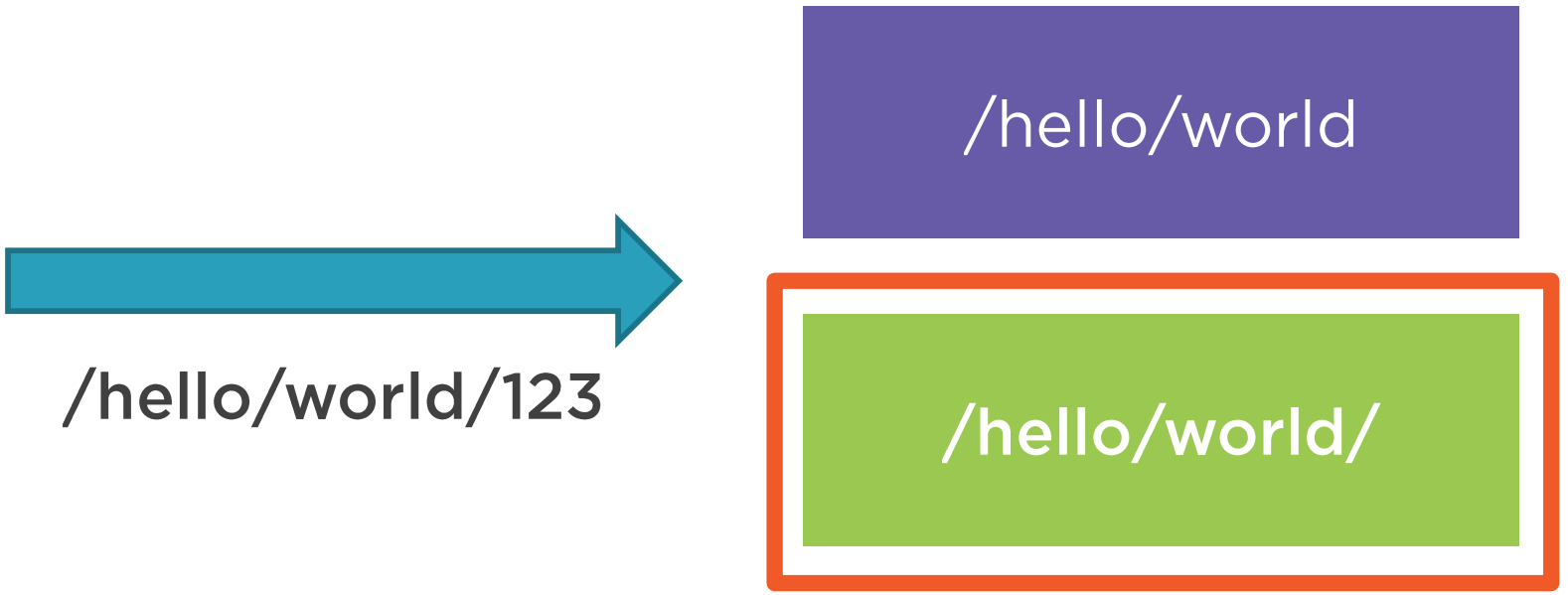
main.go

```
import “net/http”



func main() {
  foo := func(w http.ResponseWriter, _ *http.Response) {
    w.Write([]byte(“hello world”))
  }
  http.HandleFunc(“/foo”, foo)
}
```

ServeMux

HTTP Request Multiplexor

ServeMux



http.ListenAndServe

```
func ListenAndServe(addr string, handler Handler) error
```

main.go

```
import “net/http”


func main() {
  foo := func(w http.ResponseWriter, _ *http.Response) {
    w.Write([]byte(“hello world”))
  }
  http.HandleFunc(“/foo”, foo)
  err := http.ListenAndServe(“:5000”, nil)
  if err != nil {
    log.Fatal(err)
  }
}
```

main.go

```
import “net/http”

func main() {

  …

  log.Fatal(http.ListenAndServe(“:5000”, nil))
}
```

http.ListenAndServeTLS

```
func ListenAndServeTLS(addr, certFile, keyFile string, handler Handler) error
```

encoding/json

Encoding and Decoding

Marshal and Unmarshal

json.Marshal

```
func Marshal(v interface{}) ([]byte, error)
```

main.go

```
type foo struct {
  Message string
  Age int
  Name string
  surname string
}
func main() {
  data, _ := json.Marshal(&foo{“4Score”, 56, “Abe”, “Lincoln”})
  fmt.Print(string(data))
}
```

```
{"Message":“4Score",“Age":56,"Name":“Abe"}
```

json.Unmarshal

```
func Unmarshal(data []byte, v interface{}) error
```

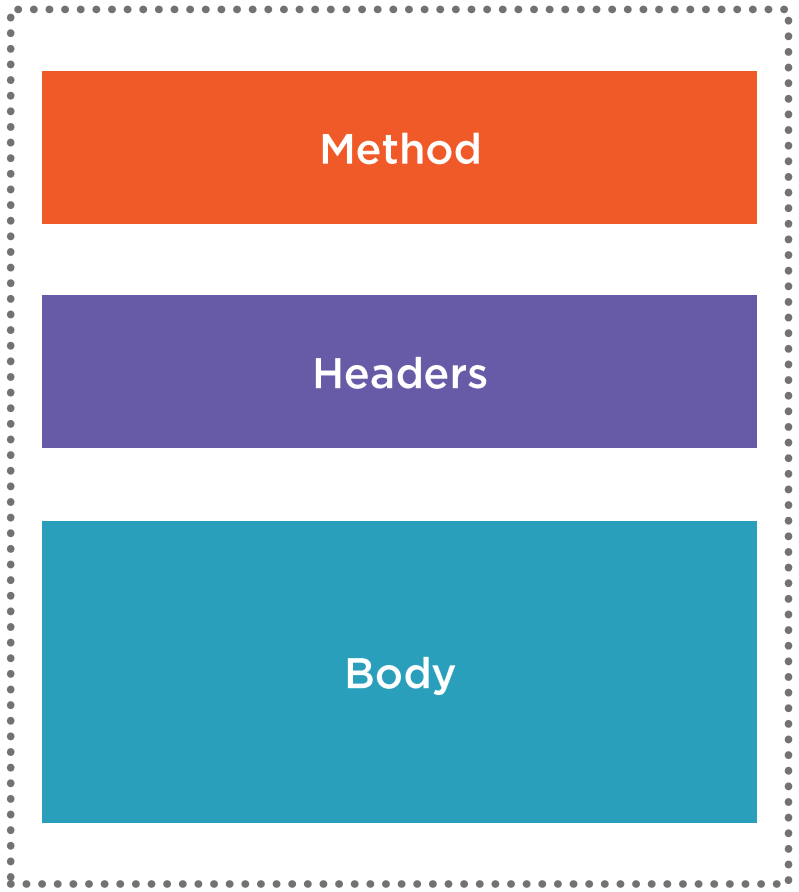
main.go

```go
func main() {
  f := foo{}
  err := json.Unmarshal([]byte(`{"Message":"4Score","Age":56,"Name":"Abe"}`), &f)
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(f.Message)
}
```

```
4Score
```

main.go

```
type foo struct {
  Message string `json: “message,omitempty”`
  Age int        `json: “age,omitempty”`
  Name string    `json: “firstName,omitempty”`
  surname string
}
```



Request.Method

- String

Request.Header

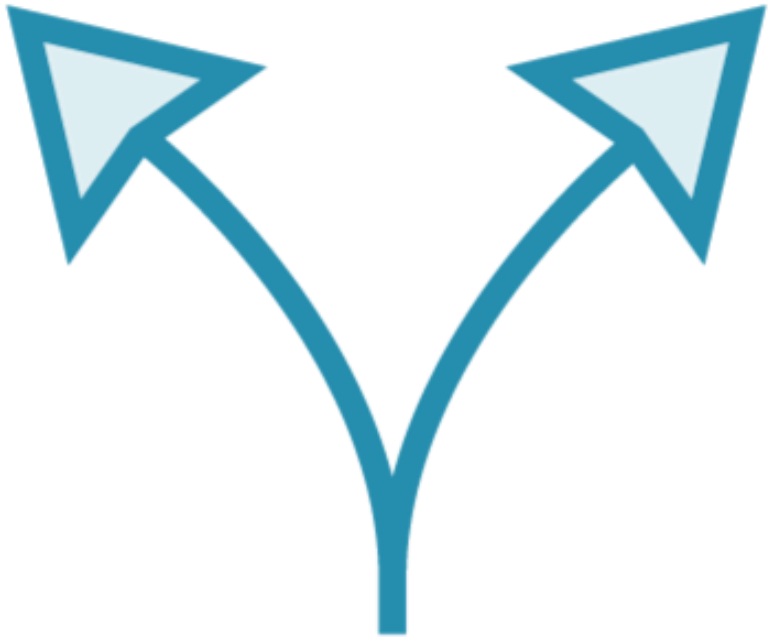
- Header (map[string][]string)

Request.Body

- io.ReadCloser
- Returns EOF when not present

/Users

GET	HTTP Method	POST
Return Users to Client	Action	Create New User



Static Routes

/products

/receipts

/users

Dynamic or Parametric Routes

/products/123

/receipts/january

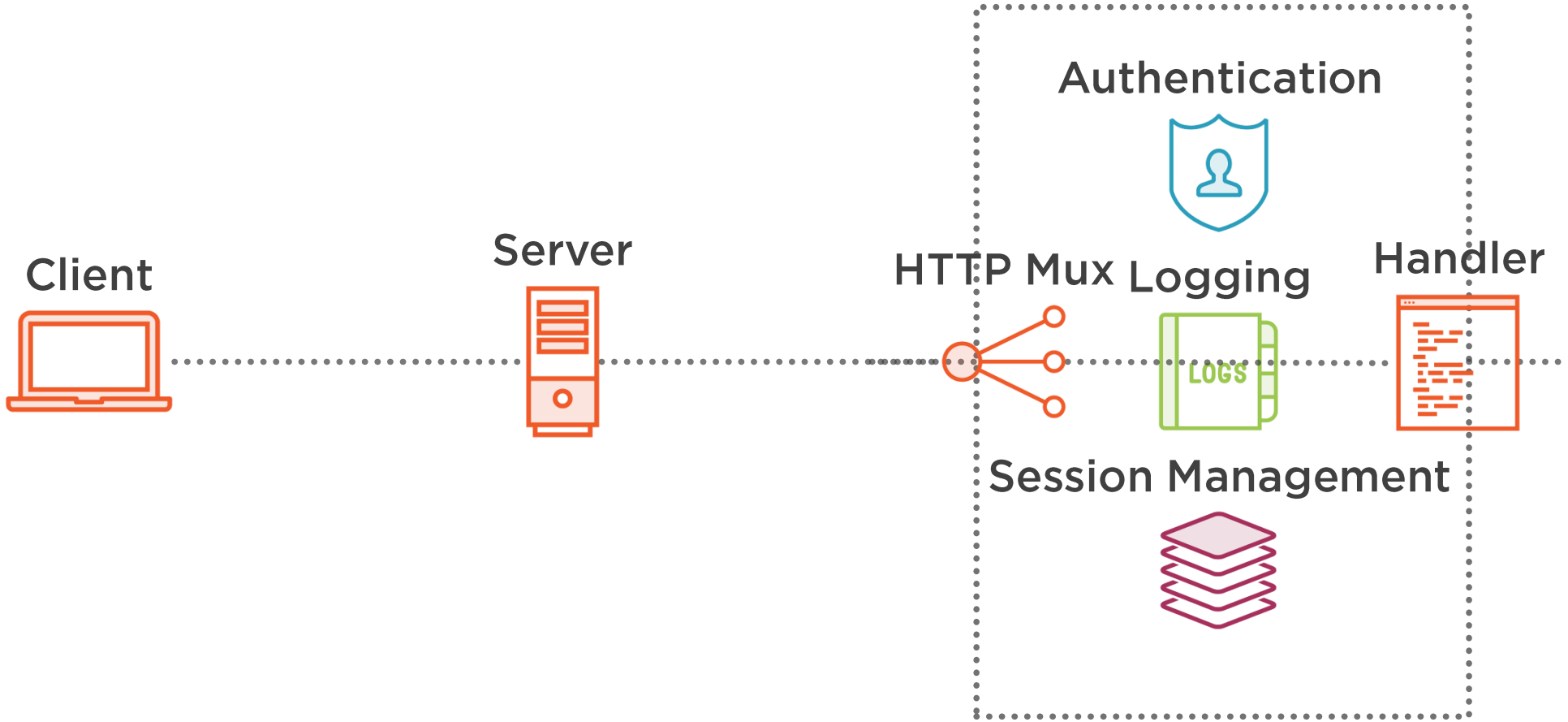
/users/bob

request.URL

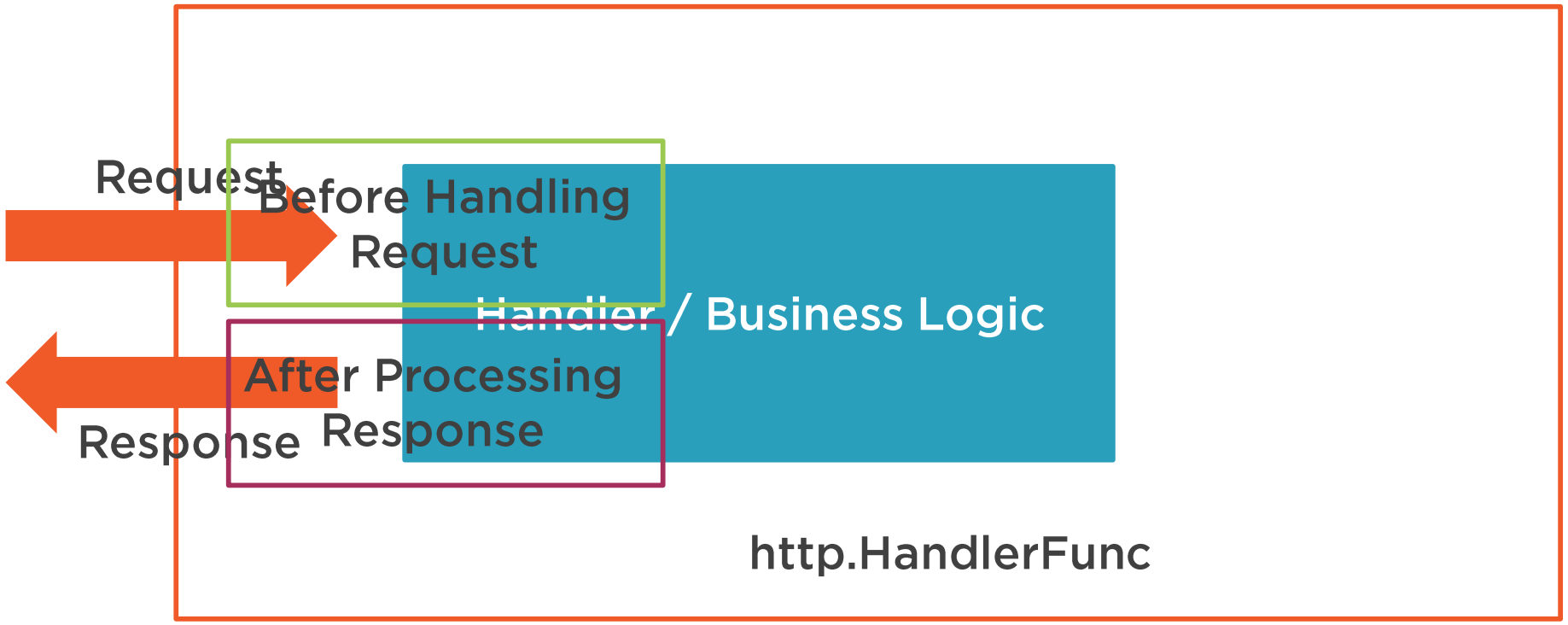
```
type URL struct {
    Scheme string
    Opaque string
    User *Userinfo
    Host string
    Path string
    RawPath string
    ForceQuery bool
    RawQuery string
    Fragment string
}
```

```
http://globomantics.com/api/products/123
```

Middleware


Client
Server
HTTP Mux
Authentication
Logging
LOGS
Handler
Session Management

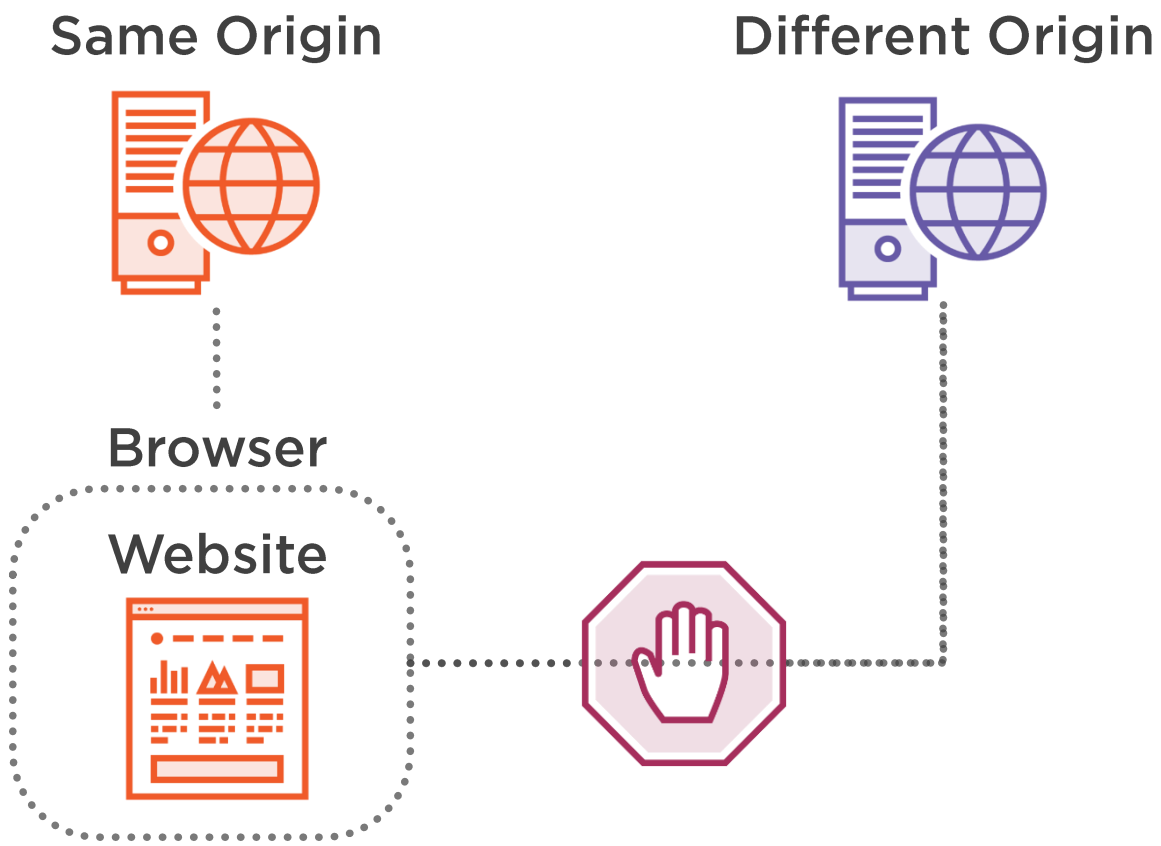
Middleware


Request
Before Handling Request
Handler / Business Logic
After Processing Response
Response
http.HandlerFunc

main.go

```go
func middlewareHandler(handler http.Handler) http.Hanlder {

  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {

  // do stuff before intended handler here

  handler.ServeHttp(w, r)

  // do stuff after intended handler here

})

 func intendedFunction(w http.ResponseWriter, r *http.Request) {
 // business logic here
 }


 func main(){
   intendedHandler := http.HandlerFunc(intendedFunction)
   http.Handle("/foo", middlewareHandler(intendedHandler))
   http.ListendAndServe(":5000", nil)
 }
```

Cross-origin Resource Sharing



http://globomantics.com

http://globomantics.com/products

http://globomantics.com/api/products/123

http://globomantics.com:8080/products

https://globomantics.com/products

http://dev.globomantics.com/dashboard

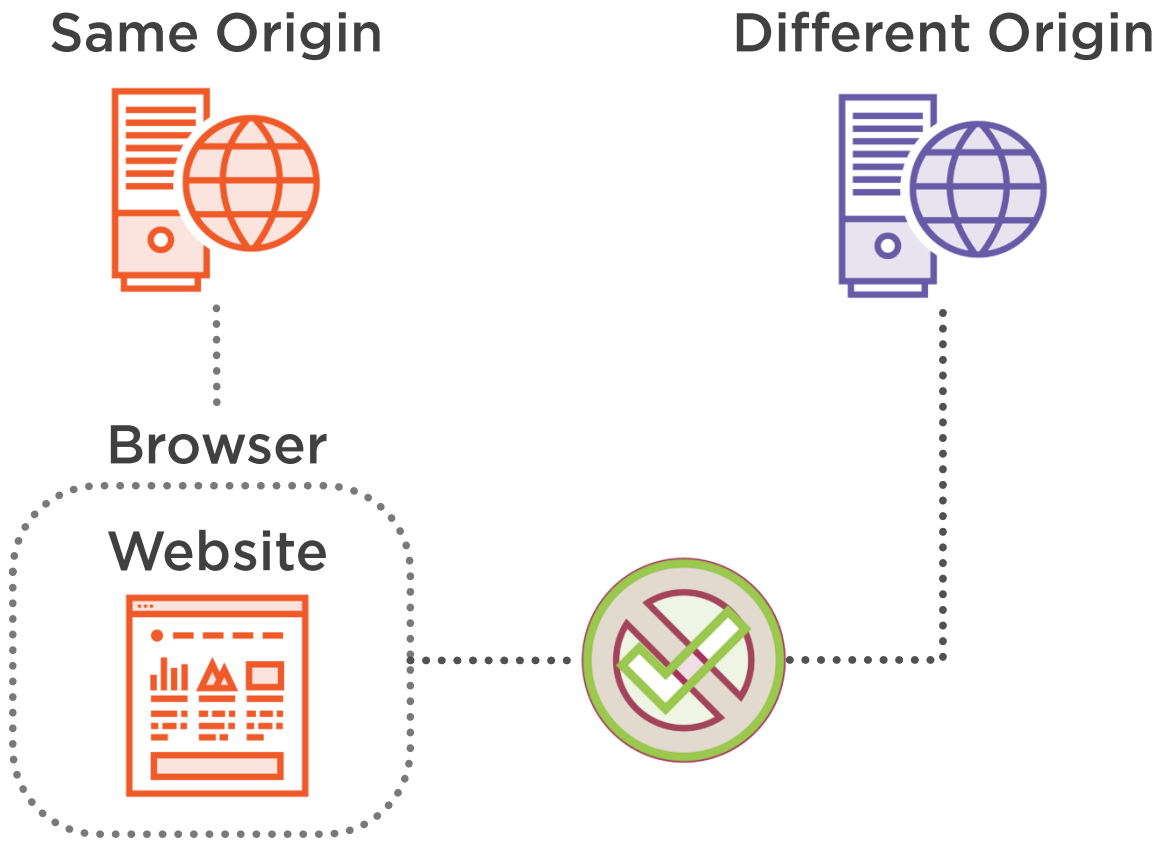
CORS Headers

Added to the http.ResponseWriter

main.go

```
w.Header().Add(“Access-Control-Allow-Origin”, “*”) // “*” allows any origin

w.Header().Add(“Access-Control-Allow-Methods”, “POST, GET, OPTIONS, PUT, DELETE”)

w.Header().Add(“Access-Control-Allow-Headers”, “Accept, Content-Type, Content-
Length, Authorization, X-CSRF-Token, Accept-Encoding”)
```

Cross-origin Resource Sharing



https://developer.mozilla.org/en-US/docs/Glossary/CORS

Demo

Organize the code into packages

Product Handler

- GET /products
- GET /products/{productID}
- POST /products
- PUT /products/{productID}
- DELETE /products/{productID}

Add CORS middleware

Summary

- Creating Basic Handlers
- Working with JSON
- Request
- URL Path
- Middleware
- CORS
- Update the Client Application