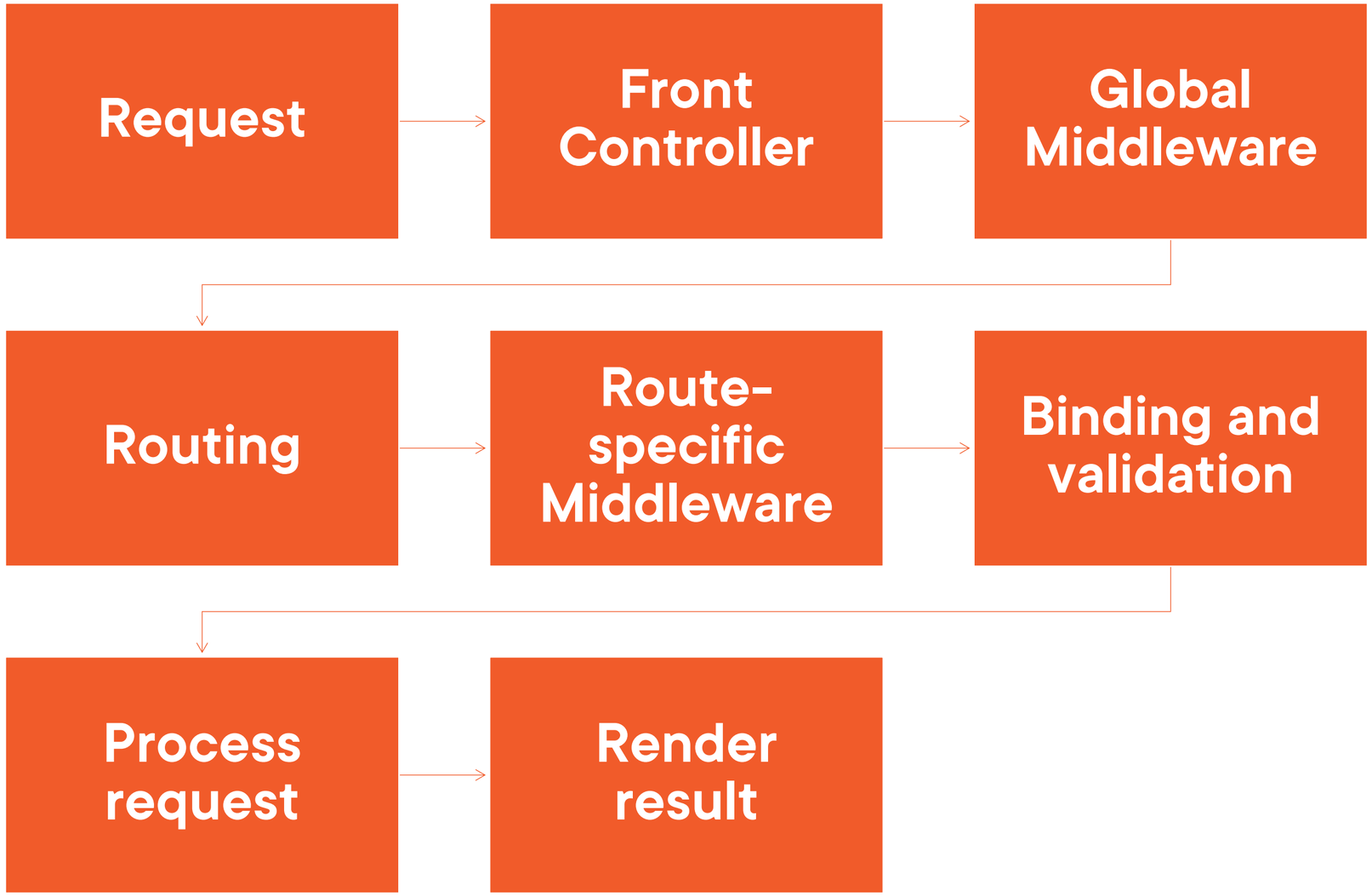
Working with Requests

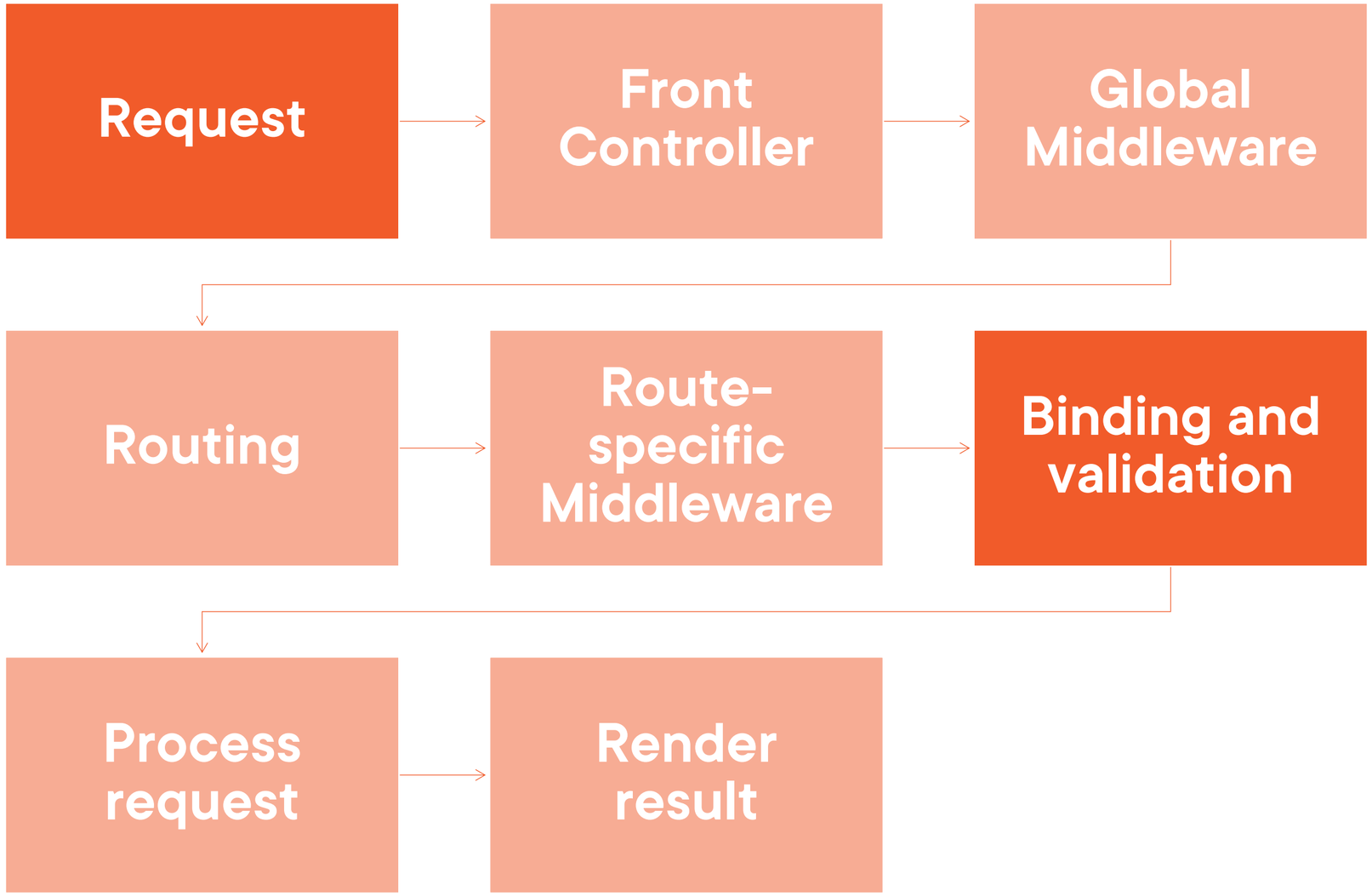
Michael Van Sickle

@vansimke

HTTP Request Pipeline



HTTP Request Pipeline



Overview

Request objects

Retrieving data and files

Data binding

Validation

Useful http.Request Members

***Request.Body**

***Request.URL**

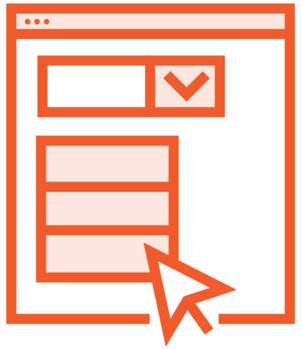
***Request.Header**

***Request.Cookies**

Retrieving Data

URL path

URL query

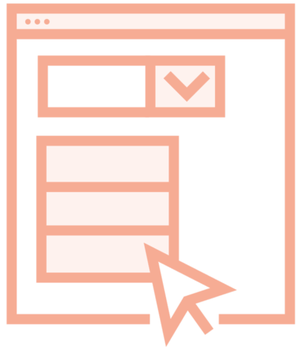


Posted forms

Retrieving Data

URL path

URL query

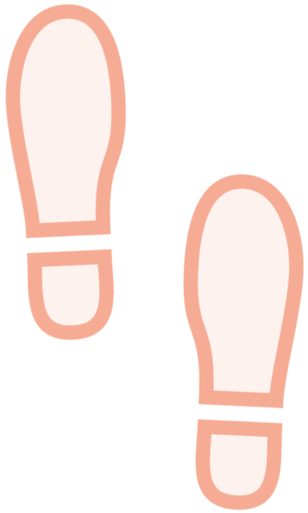


Posted forms

```
https://vacationtrack.info/users/tricia

router.GET("/users/:username",
    func(c *gin.Context) {

    c.Param("username")    // tricia

    c.Params.ByName("username")
                        // tricia

    c.Params.Get("username")
                        // tricia, true
    c.Params.Get("password")
                        // "", false
})
```

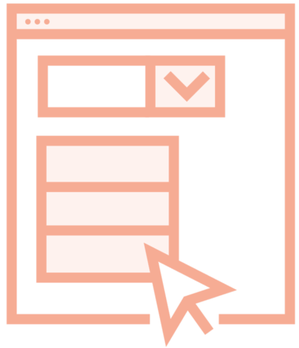
- Retrieve a parameter from URL path
- Longer way to retrieve parameter from URL path
- Retrieve parameter using comma okay pattern

Retrieving Data



URL path

URL query



Posted forms

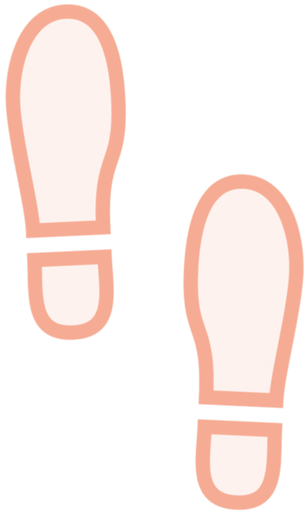
```
var c *gin.Context

c.Query(key string) string

c.QueryArray(key string) []string

c.QueryMap(key string) map[string]string

c.DefaultQuery(key, default string)
    string
```

- Retrieve first query parameter for key
- Retrieve all query parameters for key
- Retrieve query parameters encoded as map
- Retrieve query parameter if present, otherwise return “default”

URL Queries

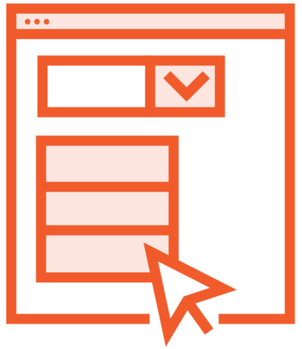
```
https://vacationtrack.info/users?username=Arthur&username=Tricia&
  scheduled[January]=0&scheduled[February]=0

router.GET(“/users”, func(c *gin.Context) {
  username := c.Query(“username”)                  // Arthur

  usernames := c.QueryArray(“username”)            // [ Arthur Tricia ]

  scheduled := c.QueryMap(“scheduled”)             // map[January:0 February:0]

  otherQuery := c.DefaultQuery(“foo”, “bar”)       // bar
})
```

Retrieving Data



URL path

URL query



Posted forms

```
var c *gin.Context

c.PostForm(key string) string

c.PostFormArray(key string) []string

c.PostFormMap(key string)
    map[string]string

c.DefaultPostForm(key, default string)
    string
```

- Retrieve first form value for key
- Retrieve all form values for key
- Retrieve form values encoded as map
- Retrieve form value if present, otherwise return “default”

Posted Forms

```
Request body
username=Arthur&username=Tricia&scheduled[January]=0&scheduled[February]=0

router.GET(“/users”, func(c *gin.Context) {
  username := c.PostForm(“username”)                    // Arthur

  usernames := c. PostFormArray(“username”)             // [ Arthur Tricia ]

  scheduled := c. PostFormMap(“scheduled”)              // map[January:0 February:0]

  otherQuery := c.DefaultPostForm(“foo”, “bar”)         // bar
})
```

```
var c *gin.Context
c.FormFile(name string)
     (*multipart.FileHeader, error)

form := c.MultipartForm()
form.Value     // map[string][]string
form.File      // map[string][]*FileHeader

var fh *FileHeader
fh.Filename
fh.Header
fh.Size
fh.Open() (File, error)

c.SaveUploadedFile(
     file *multipart.FileHeader,
     dst string) error
```

- Return uploaded file with given name
- Retrieve multipart encoded form
- Access form values
- Access uploaded files
- Name of file
- MIMEType of file
- Size of file [bytes]
- Open the file to work with it
- Save uploaded file locally

Data Binding

```
type User struct {
    FirstName string
    LastName string
}

jsonMsg := `{
    “firstname”: “Arthur”,
    “lastname”: “Dent”
}`
```

Data Binding

```
type User struct {
    FirstName string      `json:"firstname" binding:"-"`
    LastName string       `json:"lastname" binding:"-"`
}

jsonMsg := `{
    "firstname": "Arthur",
    "lastname": "Dent"
}`

var c *gin.Context
var user User
err := c.ShouldBindJSON(&user)      // err != nil if binding fails
err = c.MustBindJSON(&user)         // err != nil and context aborted if binding fails
```

```
var c *gin.Context

c.ShouldBindJSON(obj any) error

c.ShouldBindQuery (obj any) error

c.ShouldBindUri (obj any) error

c.ShouldBindHeader (obj any) error

c.ShouldBindXML (obj any) error

c.ShouldBindYAML (obj any) error

c.ShouldBind(obj any) error

c.ShouldBindWith(obj any, b

binding.Binding) error
```

- **Bind request body as JSON to object**
- **Bind query parameters to object**
- **Bind URL path parameters to object**
- **Bind header values to object**
- **Bind request body as XML to object**
- **Bind request body as YAML to object**
- **Bind request body to object, autodetect JSON or XML**
- **Bind request body with explicit binding type**

github.com/gin-gonic/gin/binding - pkg.go.dev

Binding Strategies

ShouldBind

- Return error upon failure

MustBind

- Return error upon failure
- Abort upon failure

Bind

- Alias of MustBind

```
type User struct {
  FirstName   `binding:"-"`
  LastName    `binding:"required"`
  Role
        `binding:"oneof=admin employee`
}
```

- **Optional field**
- **Required field**
- **Must be either “admin” or “employee”**

https://pkg.go.dev/github.com/go-playground/validator

Custom Validators

```
import (
  “github.com/gin-gonic/binding”
  “github.com/go-playground/validator/v10”
  <snip>
)

var myValidator validator.Func = func(f1 validator.FieldLevel) bool { ... }

func main() {
  router := gin.Default()

  if v, ok := binding.Validator.Engine().(*validator.Validate); ok {
    v.RegisterValidation(“myvalidator”, myValidator)
  }
}

type MyType struct {
  MyField   `binding:”required,myvalidator”`
}
```

Summary

- **Request objects**
- **Retrieving data and files**
- **Data binding**
- **Validation**