Hypermedia
On
Lightweight
Go
Environment under
REST

Web API Entwicklung mit GO

Gruppe: 18
6581641, Dimitri Ognev
7174298, Christoph Plusczyk

Inhalt

1. Projektvorstellung To Do Liste
2. GO Vorstellung
3. ToDo API
4. Deployment
5. Test

Projekt: To Do Liste

Full Stack Technologien:

- Web- Frontend: **React** (**Angular**)
- Webserver: **Nginx**
- Framework/ http handler: Gin, „**Gorilla-Mux**“
- Datenbank: **MongoDB**
- Backend: REST Endpoints und Handler mit **GO**
- Deployment auf **Raspberry Pi**
- Test: **Postman**

Was ist GO?

- C ähnliche Programmiersprache von Google

Besonderheiten:

- Go hat weniger Schlüsselwörter als ANSI C (25)
- GO ist weder ausdrücklich objektorientiert noch funktional (beide Paradigmen sind aber möglich)
- „Packages“ definieren Namensräume

GO Beispiel 1: Namespace, Typen

Package main

Import („project/something“)

*Func main(){ something.**W**rite(“1”, “Chris”)}*

Namespace

*//**W**rite ist außerhalb von something verfügbar*

type Bier struct {
Sorte string
Anzahl int
}

Zusammengesetzte Typen z.B. Datenstruktur („Named Tuples“ in Python)

GO Beispiel 2: Methoden

Package main Namespace

```
c := Circle{x:50, y: 50, radius:100}
c.draw()
```

Methode von Go Werte Instanz

```
func (c Circle) draw(){
        graphics.Circle(c.x, c.y, c.radius)}
```

Funkt. Wir zur Methode durch
Voranstellen des Typs auf dem die Methode basiert

GO Beispiel 3: Call by value

```
Package main
func( c Circle) setX (x int){
c.x=x}

func main(){
c:= Circle{} // mit x,y und r
c.setX(50)
Fmt.Printfn(c)}
$x:0, y:0, e:0
```

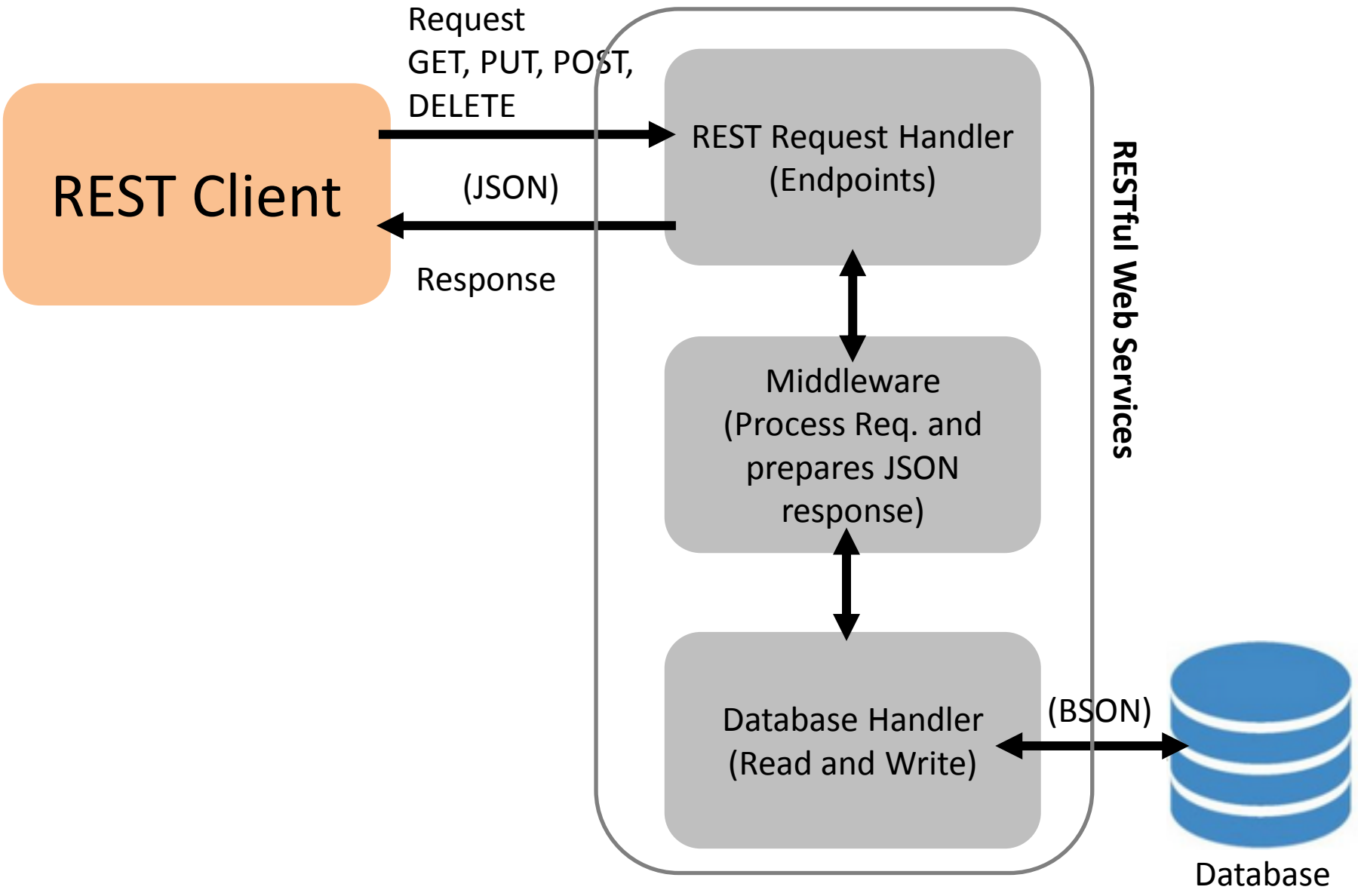
Namespace

Arbeitet auf Kopie nicht auf der c Instanz von main

Kein Seiteneffekt durch GO Call by Value Semantic

Quelle: RESTfull GO APIs by Ralf Wirdemann, 2019 Carl Hanser Verlag München

Die todo API: Three tiers



Quelle: https://www.agiratech.com/blog/building-restful-api-service-golang-using-gin-gonic

Die todo API

Frontend

Request
GET, PUT, POST,
DELETE

(JSON)

Response

REST Router
(GO, GIN, MUX)

Endpoint Handling
(GO, GIN, MUX)

Database Handler
(Read and Write
with GO)

RESTful Web Services

(BSON)

MongoDB
On AWS

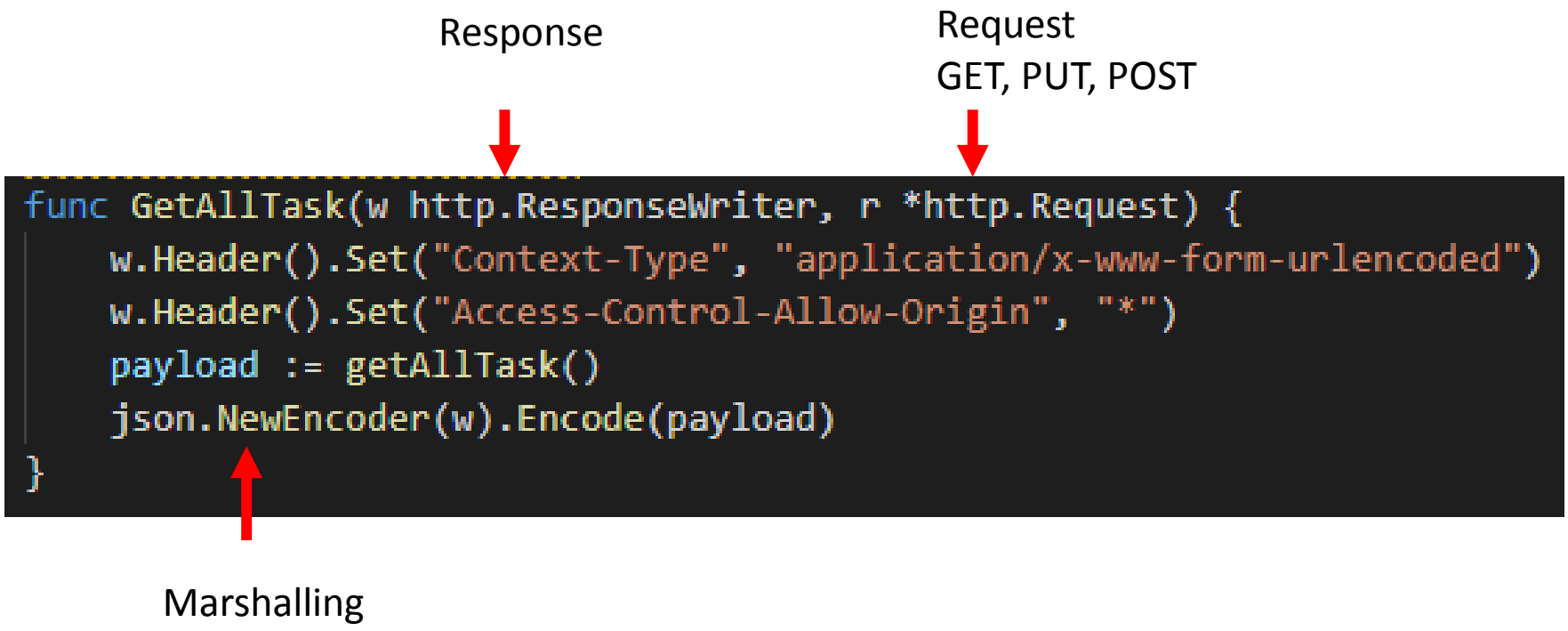
Deployment with
NGINX on Rasbian
„Cloud“

Datenstruktur

```go
package models

import "go.mongodb.org/mongo-driver/bson/primitive"

type ToDoList struct {
    ID     primitive.ObjectID `json:"_id,omitempty" bson:"_id,omitempty"`
    Task   string             `json:"task,omitempty"`
    Status bool               `json:"status,omitempty"`
}
```

Zusammengesetzte Typen z.B. Datenstruktur („Named Tuples“ in Python)

REST Endpoints: GET



REST Endpoint: POST

```go
func CreateTask(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Context-Type", "application/x-www-form-urlencoded")
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "POST")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
    var task models.ToDoList
    _ = json.NewDecoder(r.Body).Decode(&task)
    // fmt.Println(task, r.Body)
    insertOneTask(task)
    json.NewEncoder(w).Encode(task)
}
```

Marshalling

Datenstruktur

Funktionsaufruf zum Anlegen eines MongoDB Eintrages

REST Endpoint: PUT

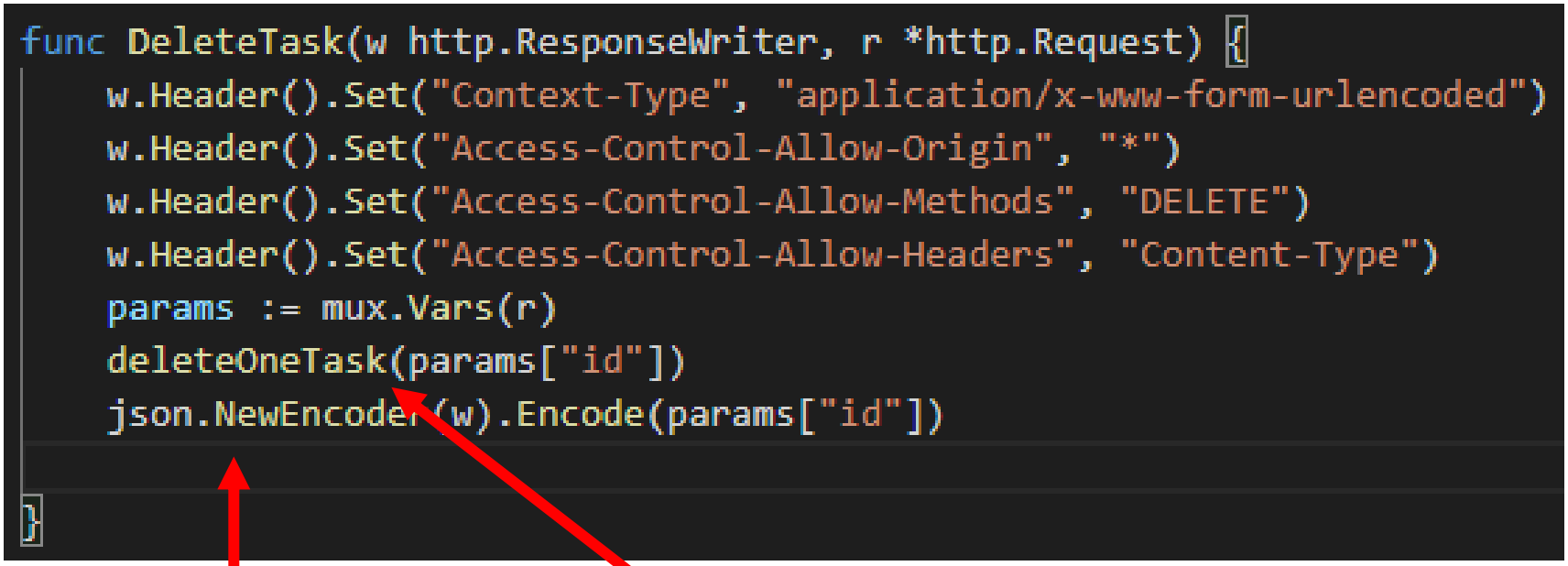
```
func TaskComplete(w http.ResponseWriter, r *http.Request) {

    w.Header().Set("Content-Type", "application/x-www-form-urlencoded")
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "PUT")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    params := mux.Vars(r)
    taskComplete(params["id"])
    json.NewEncoder(w).Encode(params["id"])
}
```

Marshalling

PUT task to true in GO Funktion mit MDB Treiber

MUX (HTTP request multiplexer): Endpoint Parameter werden zurückgegeben, geroutet

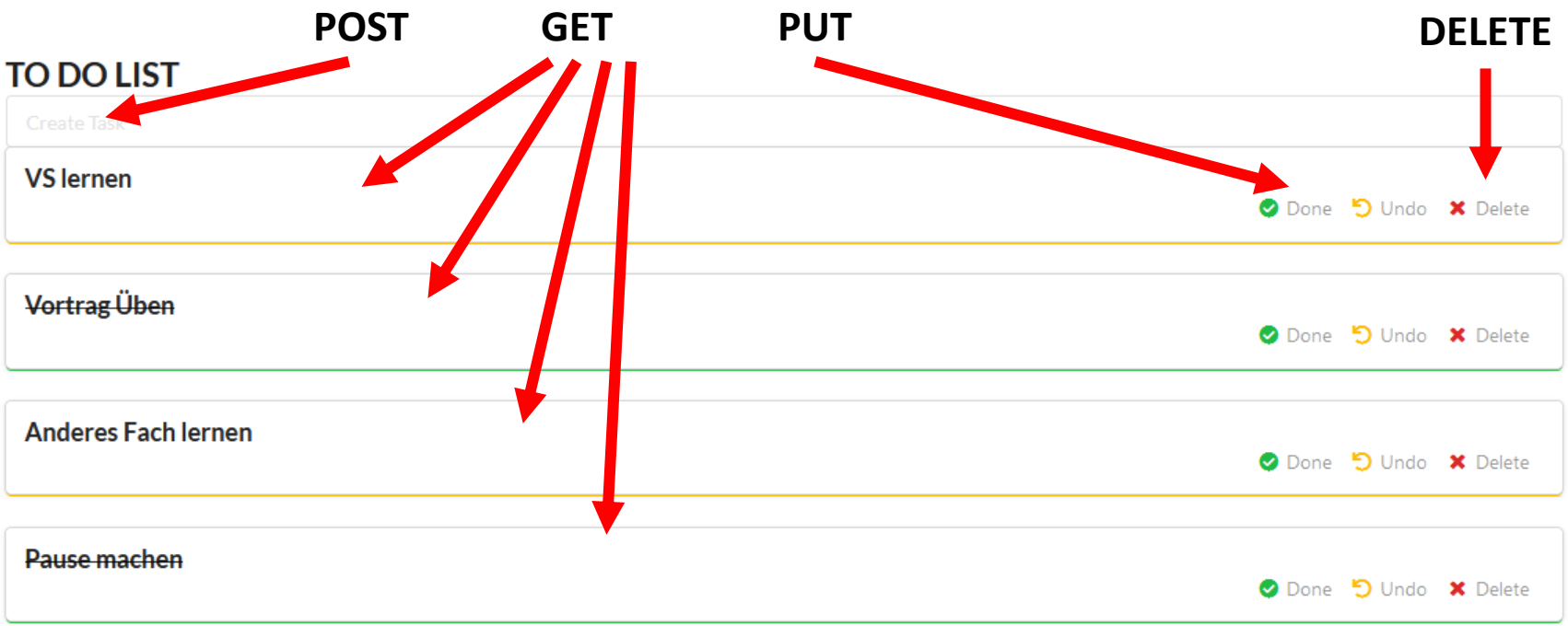
REST Endpoint: DELETE



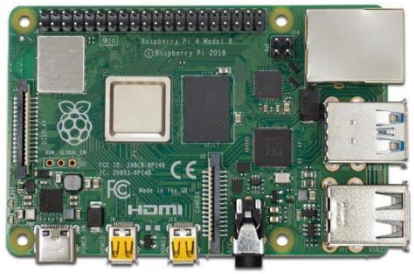
Marshalling

DELETE one task from MDB GO Funktion

Frontend


POST
GET
PUT
DELETE
TO DO LIST
Create Task
VS lernen
Done Undo Delete
~~Vortrag Üben~~
Done Undo Delete
Anderes Fach lernen
Done Undo Delete
~~Pause machen~~
Done Undo Delete

Deployment



PI Server1

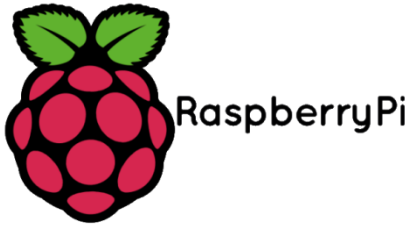
PI Server2

```
upstream www {
        server todo1.systes.net;
        server todo2.systes.net;
}
server {
        listen 80;
        server_name todo.systes.net;
        location / {
          proxy_pass http://www;
        }
}
```

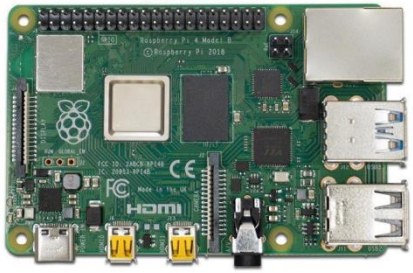
Client

NGINX load balancer (Round Robin)

Deployment



PI load balancer



NGINX Server

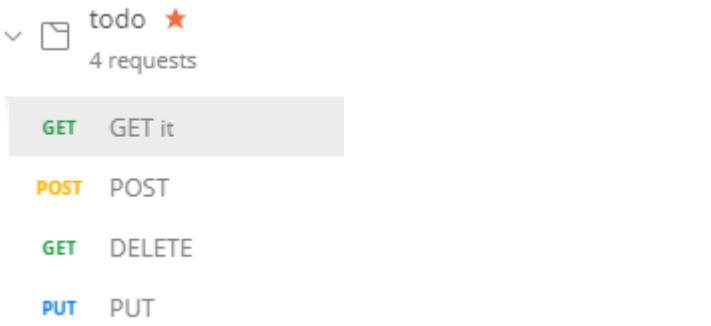
Fritzbox

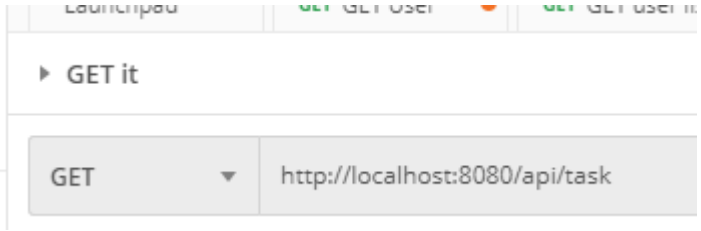
IPv6 Adresse (dynamisch)

Dynamischer DNS Anbieter

IPv6 (statisch/ 5min.)

Test


todo
4 requests
GET GET it
POST POST
GET DELETE
PUT PUT


GET it
GET
http://localhost:8080/api/task


Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
Text
1 [{"_id":"601ecbcffc65c166fde038c5","status":false,"task":"VS lernen"},{"_id":"601ecc21
2

Vielen Dank für Ihre Aufmerksamkeit!

Quellen:

GIT Hub:

- https://github.com/Chripro/VS-Abgabe

Softwareanbieter, Sticker und Symbole:

- https://golang.org/
- https://www.mongodb.com/de
- https://reactjs.org/
- https://www.nginx.com/
- https://www.postman.com/
- https://www.noip.com/
- https://www.raspberrypi.org/
- https://angular.io/