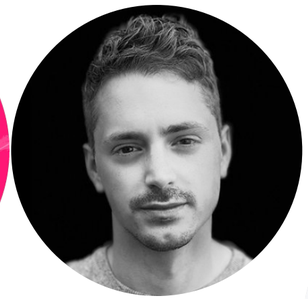
Building Go Web Services and Applications

Building a Web Service



Josh Duffney

Senior Cloud Advocate at Microsoft | Former Microsoft MVP

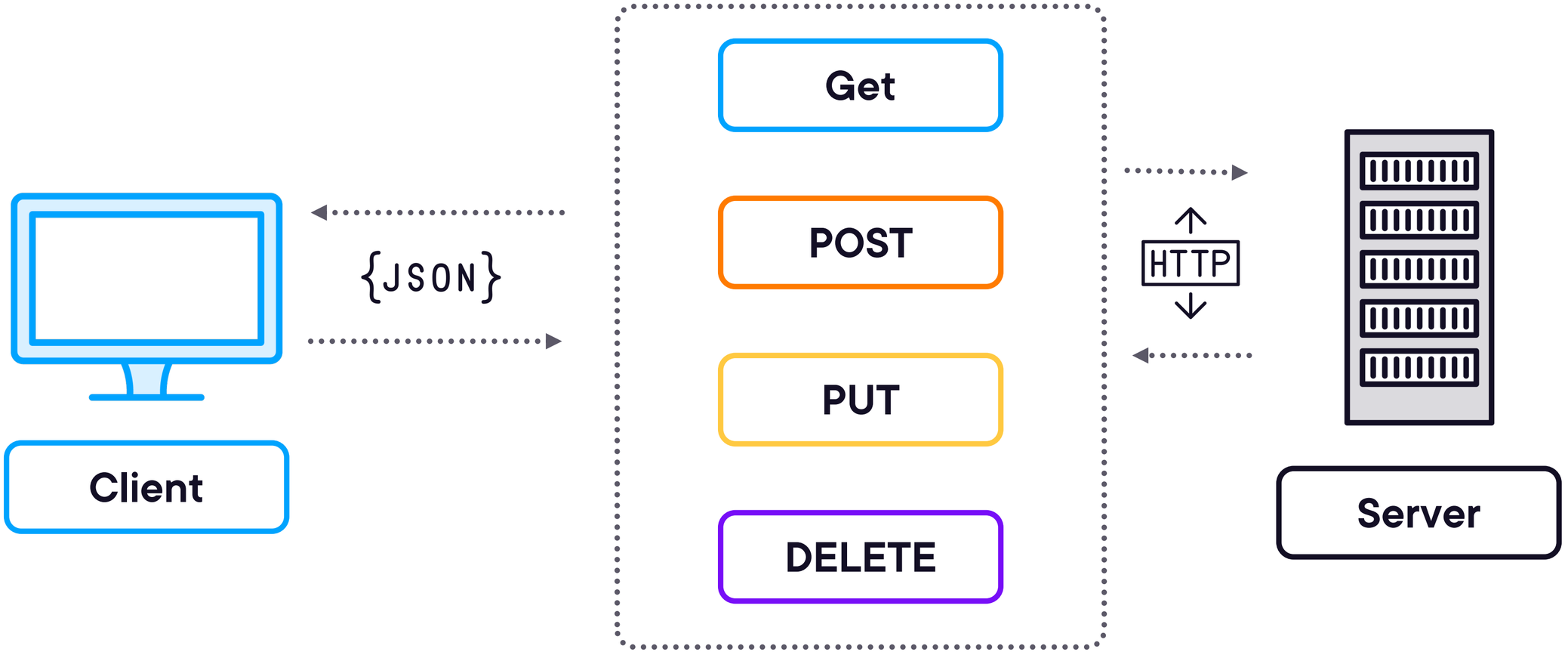
@joshduffney | www.duffney.io

Reasons for Using GO

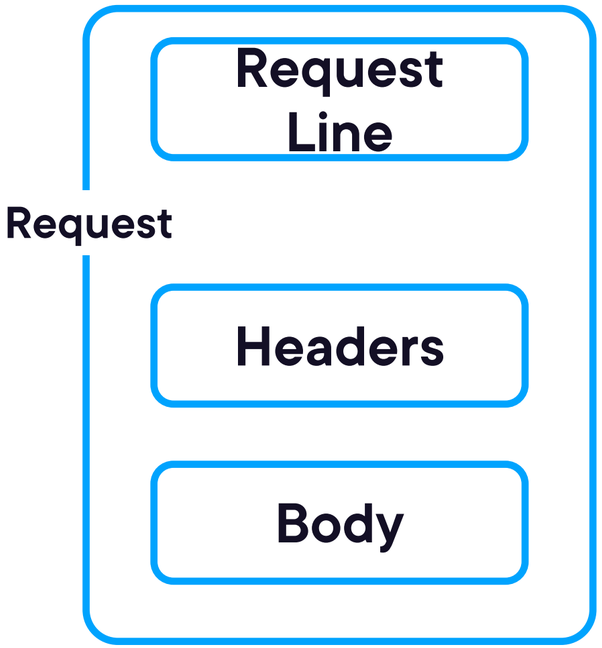
- Created by Geniuses
- Easy to Learn
- Active Open-source Community
- Built-in Concurrency

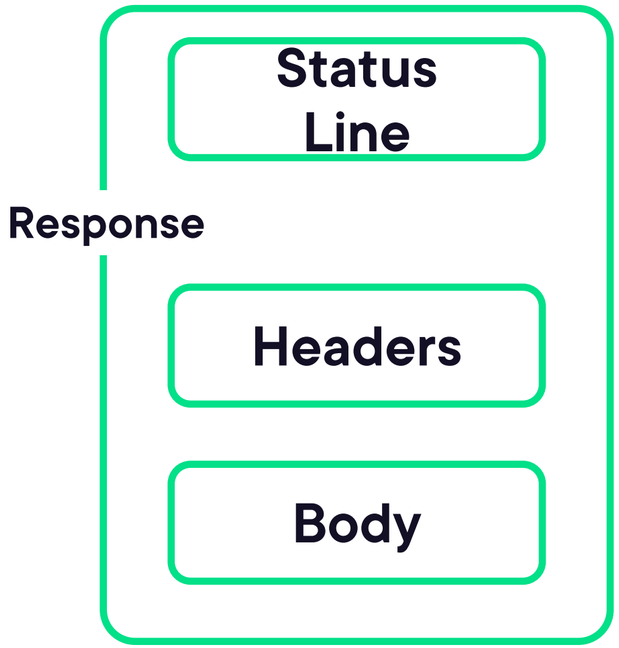
Web Services

RESTful Web Service



HTTP Requests & Reponses





Example request line:

```
> GET /hello-world HTTP/1.1
> Host: gobyexample.com
> User-Agent: curl/7.86.0
> Accept: */*
>
```

Example response line:

```
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 4025
< Connection: keep-alive
>
```

Understanding the net/http Package

```go
package main

import (
	"fmt"
	"net/http"
)

func main() {

	mux := http.NewServeMux()
	mux.HandleFunc("/", hellWorld)


	err := http.ListenAndServe(":8080", mux)
	if err != nil {
		fmt.Println(err)
	}
}

func hellWorld(w http.ResponseWriter, r *http.Request) {
	fmt.Fprint(w, "Hello, World!")
}
```

- Import the net/http package
- Create the ServeMux
- Add the handler to the ServeMux
- Start the web server
- Handle requests with a function

Sample Application: Reading List

Web Application

(Front-end) Visual user interface that visitors interact with

Written with HTML, CSS, & Go

Web Service

(Backend) RESTful API that accepts requests and supplies the web app with data from a database

Written in Go

Demo

Create a ServeMux

Add Handlers & Routes

Methods	URL Pattern
GET	/v1/healthcheck
GET, POST	/v1/books
GET, PUT, DELETE	/v1/books/

Up Next:

Working with JSON