Creating a Simple Web Server with Golang

Student's Name

Department, Institutional Affiliation

Course Name and Number

Due Date

Creating a Simple Web Server with Golang

Web servers are either software, hardware, or both working together. From the hardware perspective, a web server is a computer whose purpose is to store web server software and a website's files, such as HTML documents, images, CSS (cascading style sheets), and JavaScript files ("What is a Web Server?" n.d.). From the software perspective, a web server controls how web users access the files it hosts ("What is a Web Server?" n.d.). A web server understands HTTP (HyperText Transfer Protocol); with HTTP, the web server can understand different URLs (web addresses) and serve users with appropriate content. This paper focuses on creating a web server (the software part) using Golang (Go) language and testing it using the *curl* command-line utility.

Web servers accept HTTP requests and serve or return a response. The response can be of any format agreed upon by the developers; such formats include JSON (JavaScript Object Notation) and XML (extensible markup language). The HTTP requests consist of methods, which can be any of the following (Gole, 2018):

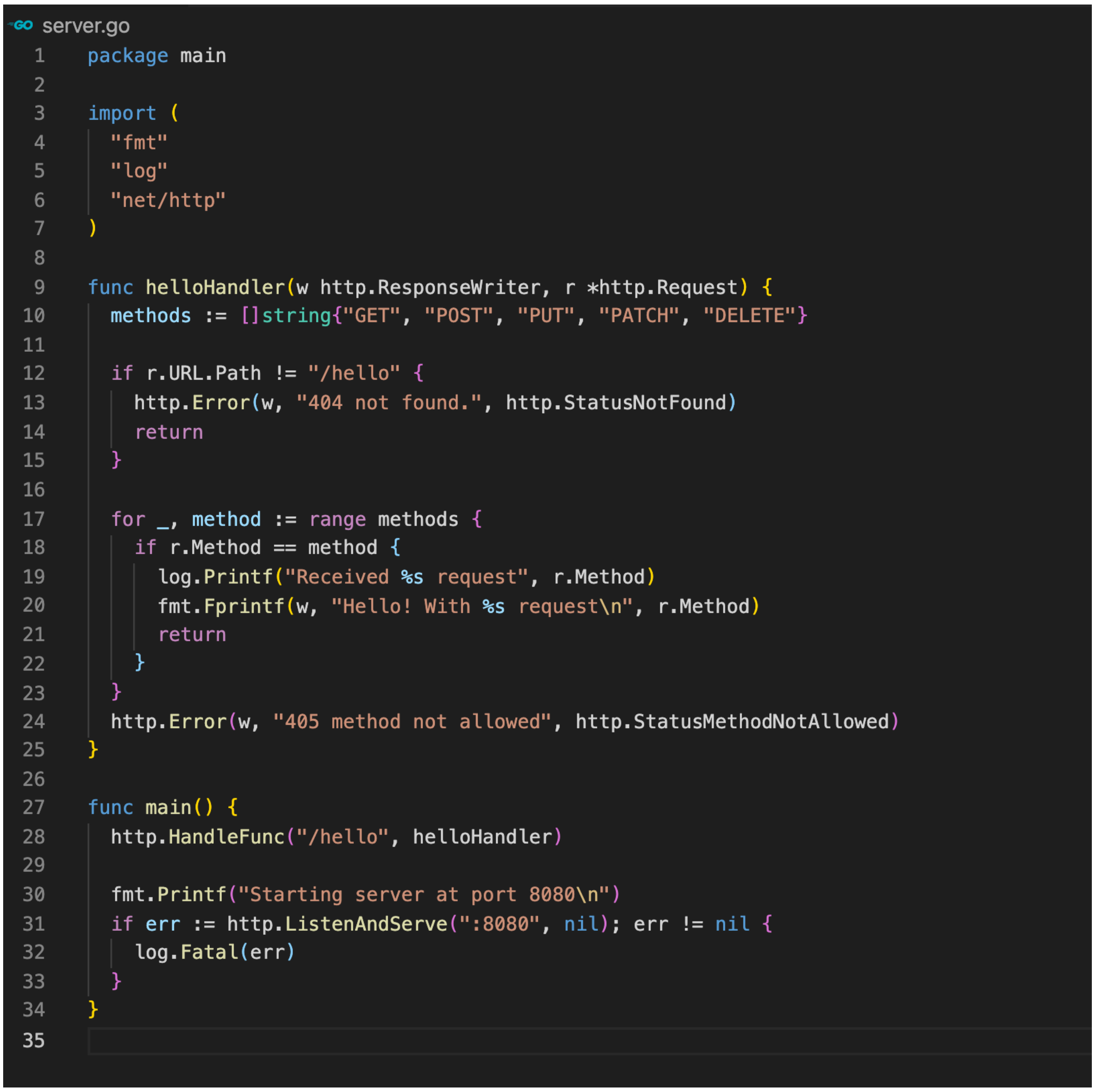
- GET – query or retrieve information from the server.
- POST – send data to the server.
- PUT – update an existing item on the server.
- PATCH – partially update an existing object on the server.
- DELETE – remove an object from the server permanently.

In Go, to build a web server, the *net/http* package is required. Figure 1 shows a simple Go web server. The web server expects requests with the methods mentioned above and returns responses, as shown in Figure 2. To start the web server, run the command *go run ./server.go*. *curl* is used to make HTTP requests.

This paper discussed what a web server is and how to create and run it in Go. It also discussed various HTTP methods and showed how to implement them in the program. Finally, it showed how to use *curl* to make requests to the web server.

Figure 1

Simple Go Web Server



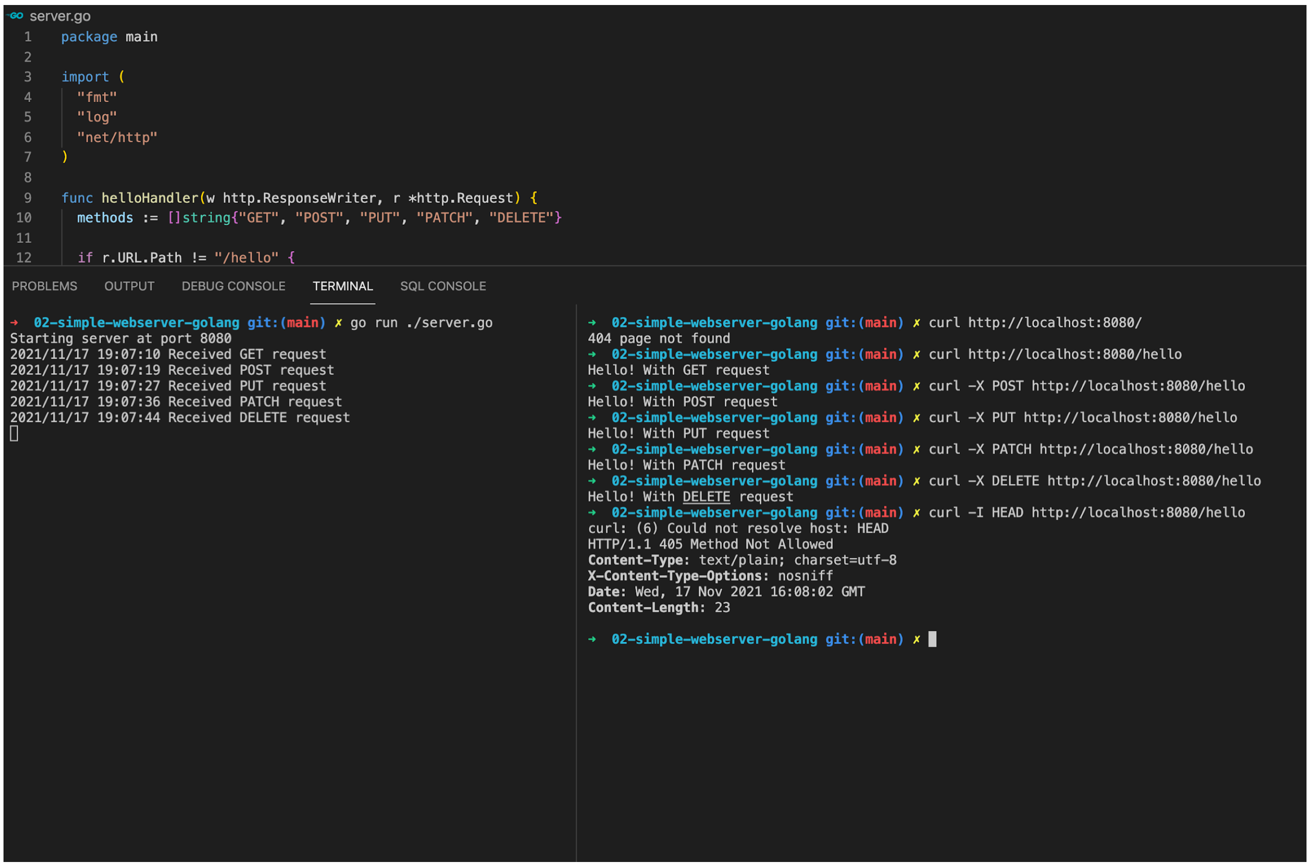
server.go

```go
package main

import (
	"fmt"
	"log"
	"net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
	methods := []string{"GET", "POST", "PUT", "PATCH", "DELETE"}

	if r.URL.Path != "/hello" {
		http.Error(w, "404 not found.", http.StatusNotFound)
		return
	}

	for _, method := range methods {
		if r.Method == method {
			log.Printf("Received %s request", r.Method)
			fmt.Fprintf(w, "Hello! With %s request\n", r.Method)
			return
		}
	}
	http.Error(w, "405 method not allowed", http.StatusMethodNotAllowed)
}

func main() {
	http.HandleFunc("/hello", helloHandler)

	fmt.Printf("Starting server at port 8080\n")
	if err := http.ListenAndServe(":8080", nil); err != nil {
		log.Fatal(err)
	}
}
```

Note. A simple web server in Go using the *net/http* package (Kamani, 2020).

Figure 2

Output of Requests Sent to the Web Server



```go
package main

import (
	"fmt"
	"log"
	"net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
	methods := []string{"GET", "POST", "PUT", "PATCH", "DELETE"}

	if r.URL.Path != "/hello" {
```

```
➜  02-simple-webserver-golang git:(main) ✗ go run ./server.go
Starting server at port 8080
2021/11/17 19:07:10 Received GET request
2021/11/17 19:07:19 Received POST request
2021/11/17 19:07:27 Received PUT request
2021/11/17 19:07:36 Received PATCH request
2021/11/17 19:07:44 Received DELETE request
```

```
➜  02-simple-webserver-golang git:(main) ✗ curl http://localhost:8080/
404 page not found
➜  02-simple-webserver-golang git:(main) ✗ curl http://localhost:8080/hello
Hello! With GET request
➜  02-simple-webserver-golang git:(main) ✗ curl -X POST http://localhost:8080/hello
Hello! With POST request
➜  02-simple-webserver-golang git:(main) ✗ curl -X PUT http://localhost:8080/hello
Hello! With PUT request
➜  02-simple-webserver-golang git:(main) ✗ curl -X PATCH http://localhost:8080/hello
Hello! With PATCH request
➜  02-simple-webserver-golang git:(main) ✗ curl -X DELETE http://localhost:8080/hello
Hello! With DELETE request
➜  02-simple-webserver-golang git:(main) ✗ curl -I HEAD http://localhost:8080/hello
curl: (6) Could not resolve host: HEAD
HTTP/1.1 405 Method Not Allowed
Content-Type: text/plain; charset=utf-8
X-Content-Type-Options: nosniff
Date: Wed, 17 Nov 2021 16:08:02 GMT
Content-Length: 23

➜  02-simple-webserver-golang git:(main) ✗
```

References

Gole, M. (2018, July 3). *A quick overview of HTTP Methods*. Medium. https://medium.com/@RP_Guests/a-quick-overview-of-http-methods-7c69e41e5aa3

Kamani, S. (2020, December 13). *Build a web application in Go (golang).* Soham Kamani. https://www.sohamkamani.com/golang/how-to-build-a-web-application/#adding-the-bird-rest-api-handlers

What is a web server? (2021, October 8). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server