Web Programming with the Go net/http Package

HTTP Model

HTTP follows a request-response model where a client (such as a web browser) sends a request to a server, and the server responds with the requested data. Each HTTP request consists of a Request Line, Headers, and an optional Body.

Go’s `net/http` package

Go’s `net/http` package provides tools to create HTTP servers and clients. It supports routing to direct incoming requests based on URL paths. It also manages requests and responses and allows for the integration of middleware and authentication.

Go’s `http.ListenAndServe` function

To start a server in Go, use the `http.ListenAndServe` function. The function takes two arguments: the port number and a handler. If the handler is `nil`, it uses the default multiplexer.

```go
err := http.ListenAndServe(":4001", nil)
if err != nil {
    log.Fatal(err)
}
```

Go’s `http.NewServeMux` function

Use `http.NewServeMux()` to create a new multiplexer in Go. This allows you to define custom routing logic and manage multiple routes.

```go
mux := http.NewServeMux()
// Define routes ...
http.ListenAndServe(":4001", mux)
```

Go's `HandleFunc` function

Go's `HandleFunc` maps URL patterns to handler functions, making routing efficient in Go web applications. Routes can include the HTTP method, host, and path.

```go
mux := http.NewServeMux()
mux.HandleFunc("/greet", handleGreet) // Route for all requests to "/greet"
http.ListenAndServe(":4001", mux)
```

Routes in web applications

Routes in web applications direct incoming requests to specific parts of the application based on the URL path. They ensure that the server processes and responds to requests correctly, enhancing the organization and navigation of the application.

GET and POST methods in Go

In Go, you can specify the GET and POST methods directly in route definitions to handle method-specific requests. This ensures that handlers respond only to the intended request methods, enhancing security.

```go
mux.HandleFunc("GET /greet", handleGetRequest) // Route for GET requests to "/greet"
mux.HandleFunc("POST /data", handlePostRequest) // Route for POST requests to "/data"
```

Go's `http.Get` function

Use Go's `http.Get` function to send a GET request to a specified URL. This function returns an `http.Response` object and a possible error, which should be handled.

```go
resp, err := http.Get(url)
if err != nil {
    log.Fatal(err)
}
```

Go’s `http.Post` function

Use Go’s `http.Post` function to send POST requests and submit data to a specified resource. The function takes three parameters: `url`, `contentType`, and `body`.

```
resp, err :=
http.Post("http://example.com/page",
"application/json",
bytes.NewBuffer(jsonData))
if err != nil {
    log.Fatal(err)
}
defer resp.Body.Close()
```

Form Handling in Go

In Go, use `ParseForm()` to parse form data sent via an HTTP POST request. Access specific form field values using `FormValue(key)`, where `key` is the name of the form field.

```
r.ParseForm() // Parse form data
title := r.FormValue("title")
content := r.FormValue("content")
```

Print   Share