



OCTOBER 7, 2024

MAKING A WEB APP WITH GO

FOR FUN AND PROFIT

WOLFLEY JR., JAMES

Outline

Ready to get started making web applications? The goal of these instructions is to give you all the tools and processes needed to start with nothing and end up with a functioning web application base. The chosen technologies for this project are Go for the backend language using the Echo library. The front end will be powered by Html and Tailwind for the CSS.

The Why

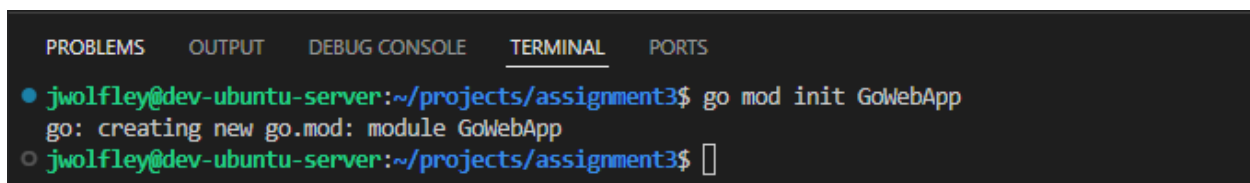
Some may ask why to choose Go as a web application backend? The reason which will exhibited by these instructions are the ease of use of which it can be done and the speed of Go. Go is a garbage collected language but still maintains a level of speed similar to that of managed memory languages. Its syntax is also very easy to read making it easy for near anyone with any coding experience the ability to learn it rather easily. These two things are not by mistake. Go was designed this way by engineers at Google while they were frustrated with C/C++ and its long compile times, poor error messages, and difficult concurrency implementations. With these as the focuses it means that this language makes a good choice for a great deal of projects including web applications.

Requirements

- Go installed on your machine. Available [here](#).
- An editor of your choice. The examples will use VS Code.
- Being somewhat comfortable with the command line

Getting Started

The first step to any new project is getting your environment setup. Go doesn't require any special IDE to work with, but it is beneficial to use the extension for your editor to help with highlighting syntax. For VS Code it is simply called Go from the "Go Team at Google". The next step is to initialize Go project with "Go mod init <name>" from the command line. Create a new directory and navigate to it then run "Go mod init GoWebApp". This creates the go.mod file which tracks the version of Go used and the dependencies.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● jwolfley@dev-ubuntu-server:~/projects/assignment3$ go mod init GoWebApp
go: creating new go.mod: module GoWebApp
○ jwolfley@dev-ubuntu-server:~/projects/assignment3$
```

Installing Dependencies

Next we will acquire all our dependencies of which there will only be two. The first and easiest is the Echo web server library. This is a simple and performant web server that is easy to use and fast. Simply run "go get github.com/labstack/echo/v4". This will use the Go package manager to install Echo v4



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
go: creating new go.mod: module GoWebApp
● jwolfley@dev-ubuntu-server:~/projects/assignment3$ go get github.com/labstack/echo/v4
go: downloading golang.org/x/net v0.24.0
go: downloading golang.org/x/sys v0.19.0
go: added github.com/labstack/echo/v4 v4.12.0
go: added github.com/labstack/gommon v0.4.2
go: added github.com/mattn/go-colorable v0.1.13
go: added github.com/mattn/go-isatty v0.0.20
go: added github.com/valyala/bytebufferpool v1.0.0
go: added github.com/valyala/fasttemplate v1.2.2
go: added golang.org/x/crypto v0.22.0
go: added golang.org/x/net v0.24.0
go: added golang.org/x/sys v0.19.0
go: added golang.org/x/text v0.14.0
○ jwolfley@dev-ubuntu-server:~/projects/assignment3$
```

You'll notice all of these things were also added to your go.mod file and a new file named go.sum was created to track these dependencies.

The next and more difficult dependency is tailwindcss. This isn't strictly necessary, but it makes it significantly easier to style the web application in comparison to handwriting all of your CSS. You can find the instructions for your respective platform at tailwindcss.com/blog/standalone-cli. Since this project does not use npm we will be using the standalone CLI. Save the correct version to your projects folder. For Linux make sure to give the file the permission to be executed.

Creating The File Structure

There will be a couple files interacted with in this project. The first will be our main Go file. Create a file named "main.go" and at the top of the file simply put the line "package main". We will come back to this file later. The next two files should be a pair of files named "input.css" and "output.css". These files will be the basis of our styling generated by tailwindcss. Inside the input.css type the following code.

```
input.css
1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;
```

This tells tailwindcss what parts we want to import and use. Following this run this command in your terminal "./tailwindcss init". This creates the configuration file we need for tailwindcss, "tailwind.config.js" which will look something like this.

```
input.css  JS tailwind.config.js  output.css
JS tailwind.config.js > ...
1  /** @type {import('tailwindcss').Config} */
2  module.exports = {
3    content: [],
4    theme: {
5      extend: {},
6    },
7    plugins: [],
8  }
9
10
```

We need to modify this a bit and add/change these 3 lines



```
input.css • JS tailwind.config.js • index.html • main.go • output.css

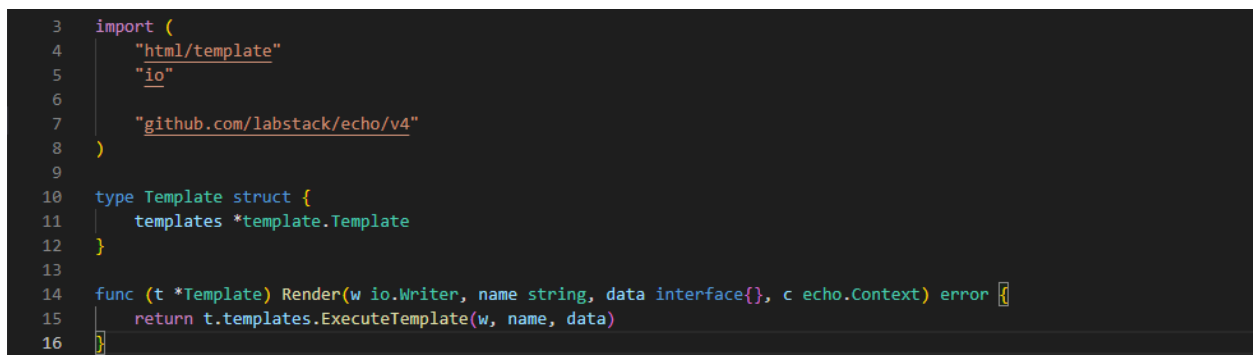
JS tailwind.config.js > ...
1  /** @type {import('tailwindcss').Config} */
2  const colors = require("tailwindcss/colors"); //1st
3  const defaultTheme = require("tailwindcss/defaultTheme"); // 2nd
4  module.exports = {
5    content: ["./views/*.html"], //3rd
6    theme: {
7      extend: {},
8    },
9    plugins: [],
10 };
11 |
```

This tells tailwindcss where our source files for our html are when it's building the output.css file.

Next create a new directory named “views” and inside make a file called “index.html”. This file will be responsible for creating our templates that Go uses to render the webpage before it is served out to the requestor.

Web Server

Creating a web server is easy but there is a bit of boilerplate we need to write first. We need a type to represent the template.Template from Go standard library and a Render helper function to implement the echo.Render interface. They look like this.



```
3  import (
4    "html/template"
5    "io"
6
7    "github.com/labstack/echo/v4"
8  )
9
10 type Template struct {
11     templates *template.Template
12 }
13
14 func (t *Template) Render(w io.Writer, name string, data interface{}, c echo.Context) error {
15     return t.templates.ExecuteTemplate(w, name, data)
16 }
```

Take special care when it automatically adds the imports it adds echo/v4 and not just echo. This render function is what essentially produces our html to be served to the client. Next, we will register our templates, create the echo server, pass in the templates, register routes, and start the echo server.

```
23 func main() {  
24     //Pre render the templates  
25     t := &Template{  
26         templates: template.Must(template.ParseGlob("./views/*.html")),  
27     }  
28     //Create the echo server, assign the template reference, and register the index route  
29     e := echo.New()  
30     e.File("/output.css", "output.css")  
31     e.Static("/images", "images")  
32     e.Renderer = t  
33     e.GET("/", Index)  
34     //Start the echo server with a logger  
35     e.Logger.Fatal(e.Start(":8080"))  
36 }
```

Along with this you need the Index helper function which handles passing the context to the renderer.

```
19 func Index(c echo.Context) error {  
20     return c.Render(http.StatusOK, "index", nil)  
21 }
```

Finally, the only thing left to do is fill out our “index.html” file in the views folder.

```
views > <> index.html > ...  
1  {{define "index"}}  
2  <!DOCTYPE html>  
3  <header>  
4      <title>ENGL 202C : Project 3</title>  
5      <meta charset="UTF-8" />  
6      <meta name="viewport" content="width=device-width, initial-scale=1" />  
7      <link rel="stylesheet" href="/css/output.css" />  
8      <div>  
9          <h4>This is a starter Go web-app</h4>  
10     </div>  
11 </header>  
12 <body></body>  
13 <footer></footer>  
14 {{end}}
```

The important bit about this document is on lines 1,2,7, and 14. The first and fourteenth line tell Go's templating engine where "index" starts and stops. The rest are important for html to be correctly rendered in the browser with our styling.

Running the Application

Now that we have written the most basic form of an application lets run it and view it in our browser. The first step is to have tailwindcss generate our CSS files. This isn't as important at the moment as we haven't styled anything but it's a good habit to get into before running the site so that any changes applied will propagate. To do this run `./tailwindcss -i input.css -o output.css`

```
jwolfley@dev-ubuntu-server:~/projects/assignment3$ ./tailwindcss -i input.css -o output.css
Rebuilding...

warn - No utility classes were detected in your source files. If this is unexpected, double-check the `content` option in your Tailwind CSS configuration.
warn - https://tailwindcss.com/docs/content-configuration

Done in 202ms.
jwolfley@dev-ubuntu-server:~/projects/assignment3$
```

This error is fine for now, this is only because we have not used any of the styles yet. After that we can have go run our application with `go run .` This command compiles and runs the web application. When it's running you should see this in your terminal.

```
jwolfley@dev-ubuntu-server:~/projects/assignment3$ go run .

      _ _ _ _ _
     /___/___/___\
    /___/___/___/___\ v4.12.0
   /___/___/___/___/___\
  High performance, minimalist Go web framework
  https://echo.labstack.com

  _____o/_____
             o\
  http server started on [::]:8080
```

This means Echo is running and will display any requests made to the server in the log. To navigate to your freshly made web app open a browser and go to <http://localhost:8080> You should see this in your browser.

This is a starter Go web-app

Conclusion

In the end this is the most basic of web applications you could possibly make. The lessons learned here are still invaluable. You have now know how to run the web server, register new routes, create templates, and integrate tailwindcss for easy styling. With this as a starting point it is easy to expand and add more features from here. Often times the hardest part of learning how these technologies work is starting. From here feel free to play with the templates, try to get another page to load and link the two together. Treat this platform as a sandbox for learning. The full source code for this project as well as the completed version can be found on my github – <https://www.github.com/James-Wolfley/assignment3>. A live version of this site which will be a web version of these instructions finished with this base is available here <http://eng.wolfleymedia.com/>