



Portable Go apps with a web interface



David Stotijn

Freelance Software Engineer

[@dstotijn](#)

Go meetup @ Stream (Amsterdam)

May 31, 2022

Typical approach

- Go program runs HTTP server.
- Serves web content (dynamic or static) and API routes.
- User is prompted to visit ``http://localhost:8080``.
- I/O happens via HTTP requests between client (browser) and server (Go program).

The `embed` package

- Allows you to embed files at *compile time* with the `//go:embed` directive.
- Before Go 1.16, `go-bindata`, `statik`, `go.rice` et al provided this behaviour.
- No need to read files from the host OS's file system at *runtime*.
- The `embed.FS` type implements `fs.FS`.

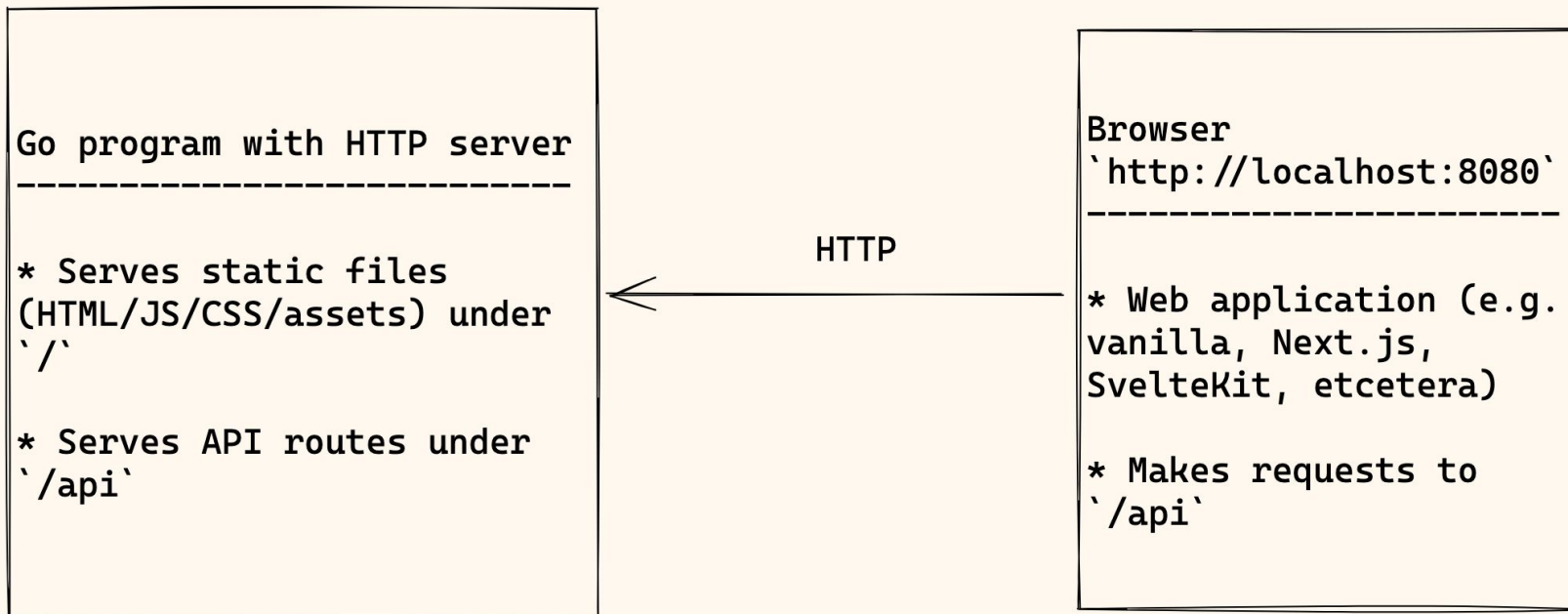
The `fs.FS` interface

- Go 1.16 also introduced the `io/fs` package with `fs.FS` interface; an abstraction for read-only trees of files:

```
type FS interface {  
    // Open opens the named file.  
    Open(name string) (File, error)  
}
```

- The `fs.FS` interface is used in `embed`, `net/http`, `text/template`, and `html/template`.

Go app with a “SPA” frontend



Example

- Embed tree of files (e.g. the static export of a Next.js app):
`//go:embed all:nextjs/dist`
`var nextFS embed.FS`
- Root at a subtree that contains our static export:
`distFS, err := fs.Sub(nextFS, "nextjs/dist")`
- Serve contents via ``http.FileServer``, which returns an ``http.Handler``.
- Serve API routes separately (e.g. under ``/api``). The web UI uses these routes for fetching/sending data.

```
package main
```

```
import (  
    "embed"  
    "io/fs"  
    "log"  
    "net/http"  
)
```

```
//go:embed all:nextjs/dist  
var nextFS embed.FS
```

```
func main() {  
    // Root at the `dist` folder generated by the Next.js app.  
    distFS, err := fs.Sub(nextFS, "nextjs/dist")  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    // The static Next.js app will be served under `/.`.  
    http.Handle("/", http.FileServer(http.FS(distFS)))  
  
    // The API will be served under `/api`.  
    http.HandleFunc("/api", apiHandler)  
  
    // Start HTTP server at :8080.  
    log.Println("Starting HTTP server at http://localhost:8080 ...")  
    log.Fatal(http.ListenAndServe(":8080", nil))  
}
```

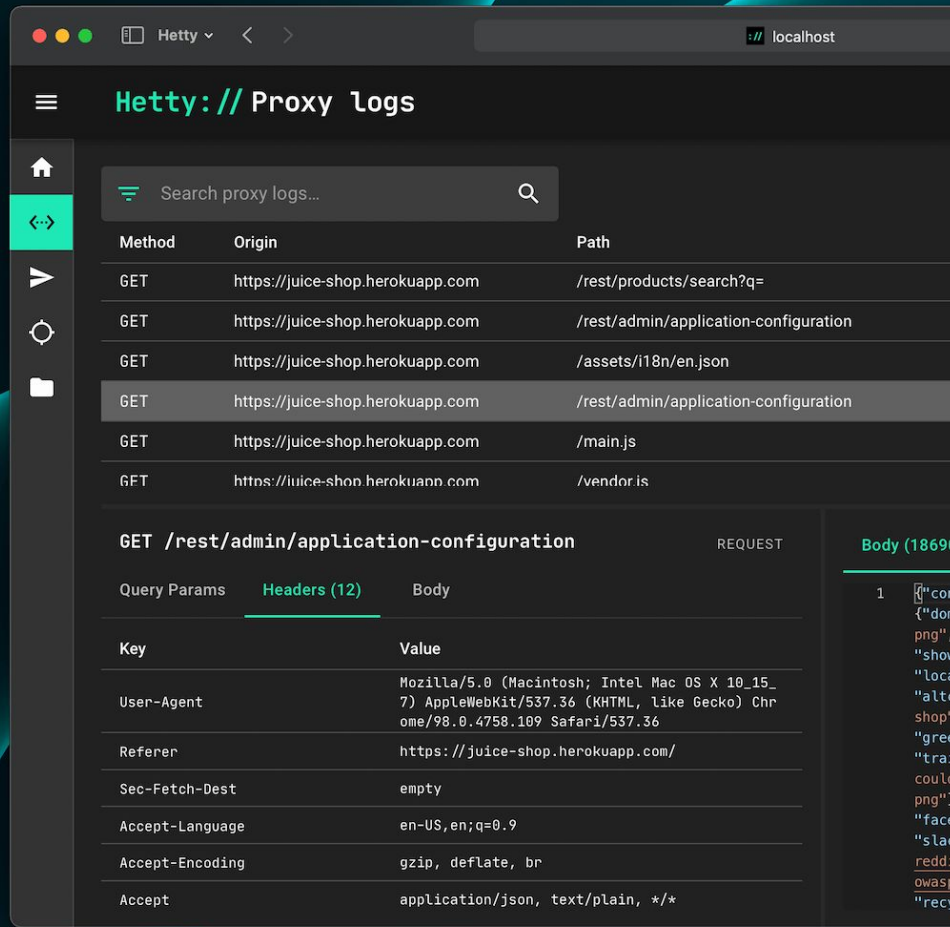
```
func apiHandler(rw http.ResponseWriter, r *http.Request) {  
    // TODO: Handle API requests ...  
}
```

Example: Hetty

“An HTTP toolkit for security research.”

- Next.js (static export) frontend
- GraphQL server in Go
- Apollo Client (GraphQL) on the frontend

 <https://hetty.xyz>



The screenshot shows the Hetty web interface running on localhost. The main panel displays a list of proxy logs with columns for Method, Origin, and Path. The selected log is a GET request to /rest/admin/application-configuration. Below the log list, the details for this request are shown, including the Headers tab which lists various HTTP headers like User-Agent, Referer, and Accept.

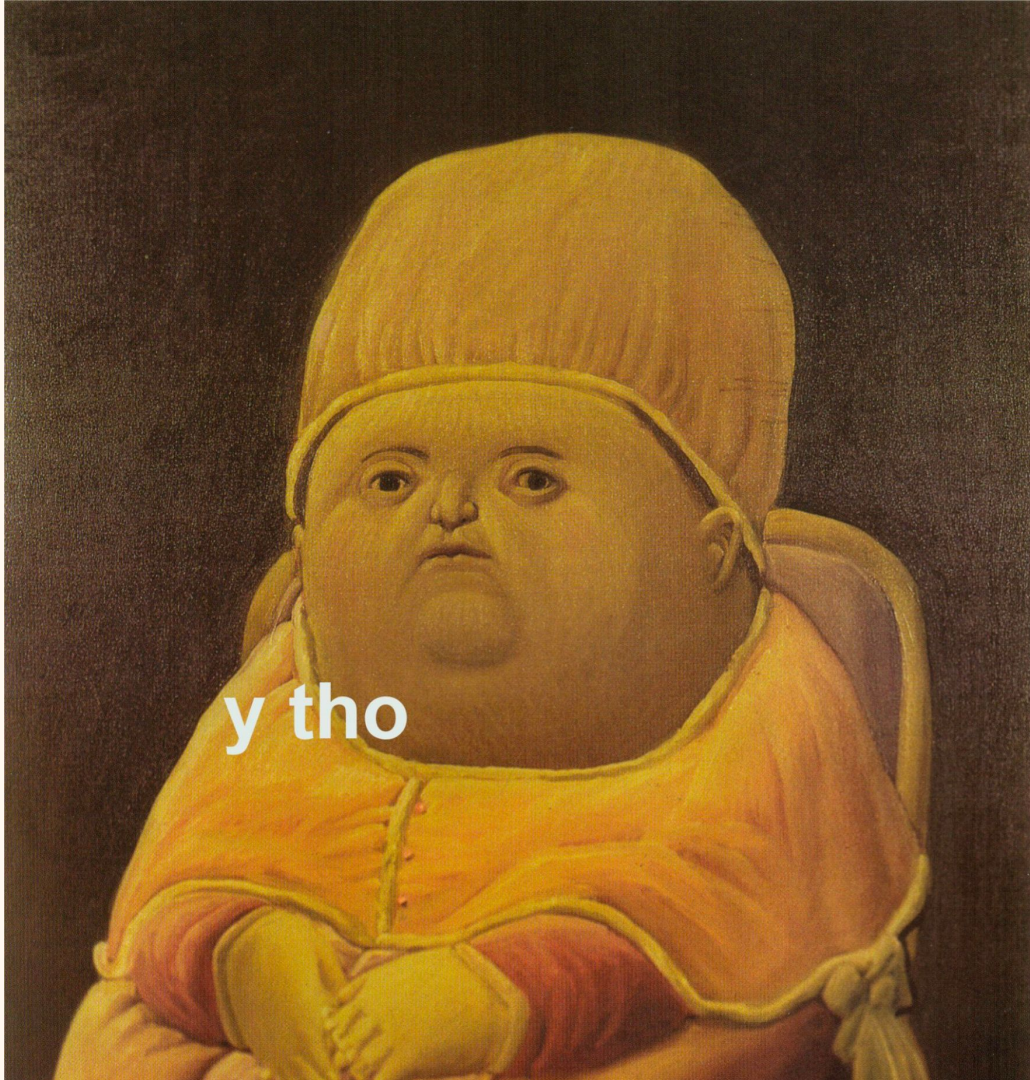
Method	Origin	Path
GET	https://juice-shop.herokuapp.com	/rest/products/search?q=
GET	https://juice-shop.herokuapp.com	/rest/admin/application-configuration
GET	https://juice-shop.herokuapp.com	/assets/i18n/en.json
GET	https://juice-shop.herokuapp.com	/rest/admin/application-configuration
GET	https://juice-shop.herokuapp.com	/main.js
GET	https://juice-shop.herokuapp.com	/vendor.js

GET /rest/admin/application-configuration		REQUEST														
Query Params	Headers (12)	Body														
<table border="1"><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>User-Agent</td><td>Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36</td></tr><tr><td>Referer</td><td>https://juice-shop.herokuapp.com/</td></tr><tr><td>Sec-Fetch-Dest</td><td>empty</td></tr><tr><td>Accept-Language</td><td>en-US,en;q=0.9</td></tr><tr><td>Accept-Encoding</td><td>gzip, deflate, br</td></tr><tr><td>Accept</td><td>application/json, text/plain, */*</td></tr></tbody></table>		Key	Value	User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36	Referer	https://juice-shop.herokuapp.com/	Sec-Fetch-Dest	empty	Accept-Language	en-US,en;q=0.9	Accept-Encoding	gzip, deflate, br	Accept	application/json, text/plain, */*	
Key	Value															
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36															
Referer	https://juice-shop.herokuapp.com/															
Sec-Fetch-Dest	empty															
Accept-Language	en-US,en;q=0.9															
Accept-Encoding	gzip, deflate, br															
Accept	application/json, text/plain, */*															

Do you really need this?

- When does a web UI make sense?
- What about a terminal UI (TUI)?
- Why not a native GUI (Qt, Fyne)?

y tho



Wails – Taking this a step further

- Uses webkit. More portability at the cost of more complex toolchain and dependencies.
- Inter-process communication (IPC) between Go and JS, bindings for event handling. “RPC” style I/O.
- More control over window management.
- Native dialog boxes and menus.

 <https://github.com/wailsapp/wails>

Thank you!

🌐 Slides: <https://tinyurl.com/portable-go-web-ui>

🌐 Article: <https://v0x.nl/articles/portable-apps-go-nextjs>

🌐 Example: <https://github.com/dstotijn/golang-nextjs-portable>

👉 **I'm available for contract work!**