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

```
Go program with HTTP server
---------------------------

* Serves static files
(HTML/JS/CSS/assets) under
`/`

* Serves API routes under
`/api`
```

HTTP

```
Browser
`http://localhost:8080`
-----------------------

* Web application (e.g.
vanilla, Next.js,
SvelteKit, etcetera)

* Makes requests to
`/api`
```

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.

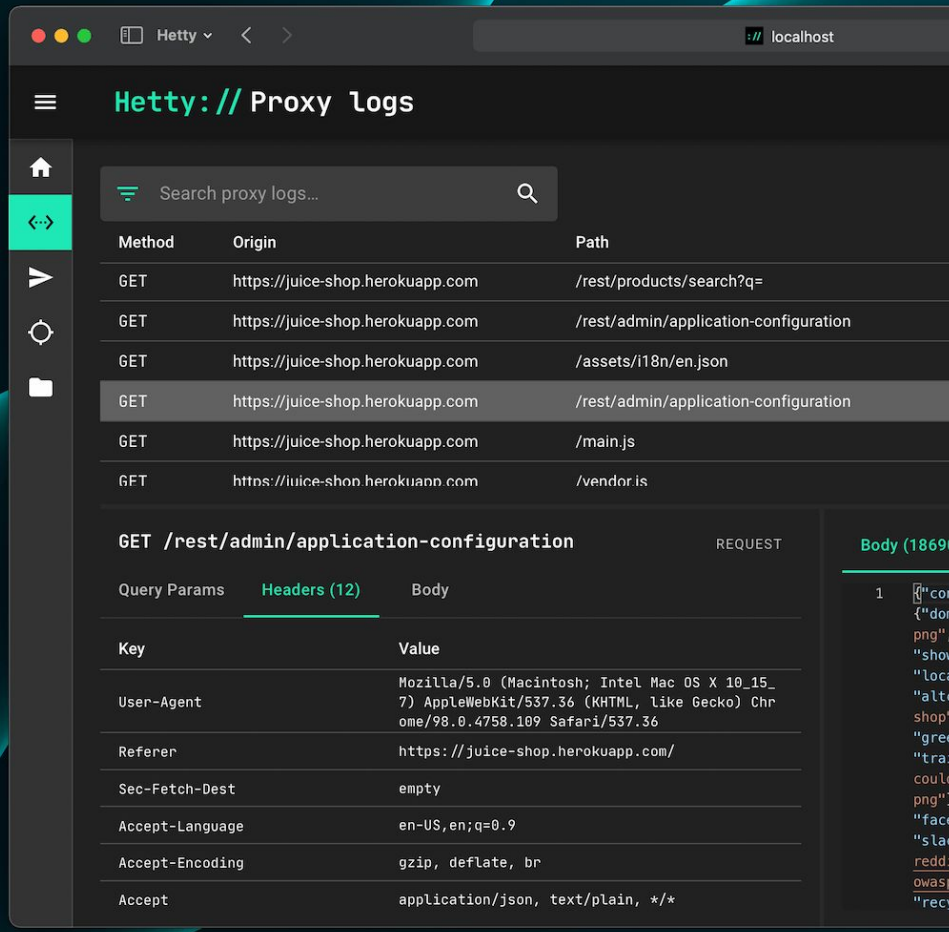
```go
package main

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.”

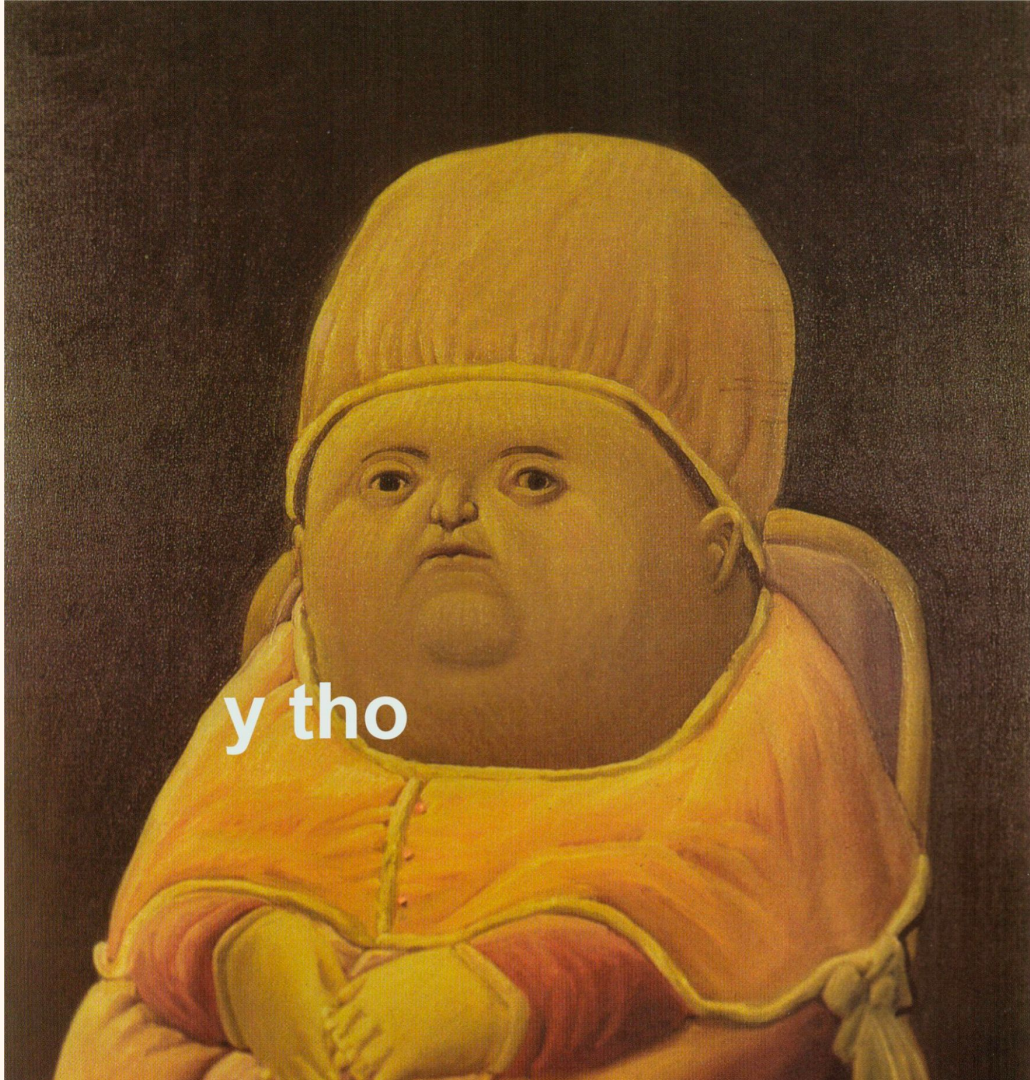
- Next.js (static export) frontend
- GraphQL server in Go
- Apollo Client (GraphQL) on the frontend

🌐 https://hetty.xyz



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)?



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!**