GO Live Reload with Automatic Page Refresh

https://github.com/dmh2000/go-live-reload

Introduction

If you are coming from the React world of web app development, you probably use 'create-react-app','vite' or some other tool to scaffold out your application. One feature that these provide is live reload of the app when you change something in the source code. It really speeds up development.

In the Go world, you can get live reload of a web app using cosmtrek/air or one of a few other tools : https://techinscribed.com/5-ways-to-live-reloading-go-applications/

THERE IS ONE CATCH WITH GO: In the React environment, when you change code, the server will restart, **AND** the changes show up immediately on the browser. This is possible because a React app has a bunch of JavaScript running in the app that handle it. Its baked into the React library that is part of the app.

On the other hand, the go tools such as Air will restart your server when source code changes, but they don't force the browser to refresh the current page. So as far as I know with any of the go approaches, you have to manually refresh the page (F5) or possibly have it refresh itself with a periodic timer (horrible?).

There is a solution that seems to work for me. By adding a bit of code, you can have REAL live reload with page refresh on code change.

Approach

1.Install cosmtrek/air which will be used to run the go web server in the conventional manner. **air** will restart the web server when the source is changed. **air** does not itself force a reload of the web page in the browser.

2. In the main web server code, import github.com/dmh2000/autoreload.
3. Add a call to autoreload.Start in your main web server. This creates a websocket endpoint that the reloader in the web pages connect. Call autoreload.Start before starting the web server. See example/main.go. Also add the autoreload.Handler to your http server mux. You can set the url to anything that works for your system.

```
// from autoreload package
func Start(url string, port int,origin string)...

func Handler(w http.ResponseWriter, r *http.Request)
```

```
    const reloadPort = 8080                  // port that the reload
websocket server listens on
    const reloadUrl = "/reload"              // url that the reload
websocket server listens on
```

```
    const origin =  "http://localhost:8001" // used for origin check in
websocket upgrade
    // start the autoreload server which will spawn a goroutine that
listens for websocket connections
    autoreload.Start(reloadUrl,reloadPort, origin)
    // add autoreload handler to the mux. url should match the snippet in
the html file
    http.HandleFunc("/reload/reloader.js", autoreload.Handler)
```

4. Add a script reference to reloader.js into your web pages in the Head section. This script creates a simple WebSocket server that listens for a connection from a WebSocket client. See example/views/index.html. The purpose of this script is to open a WebSocket connection to the autoreload function in the web server

```
<script src="/reload/reloader.js"></script>
```

What Does This Do?

Startup

- Start the Go web server using Air
  - .air.toml is configured to restart the web server on code change.
  - When the web app first loads, it will enable the WebSocket script which will wait for a connection from the client.
- Once the WebSocket endpoints are connected up, there are no messages sent back and forth. There is no extra load on the application. Both ends are passively sitting waiting for a disconnect.

Source Code Change

- Developer changes the code, either the go source or the page templates.
- In a terminal, Air will restart the Go web server. This does not by itself cause a reload. It will however break the WebSocket connection with the web page.
- In the browser, the WebSocket script in the web page detects a disconnect event.
  - After a slight delay, the script forces a web page reload and the changes show up.

Once the web page reloads and the client restarts, they will reconnect and wait for the next code change.

Note: The setTimeout delay in the scripts makes the web page refresh a little smoother. Without a delay, the browser might momentarily flash a 'site not found' message and then load the web page. It will all work without a delay, but with it you will just see the changes appear. You can tune the delay to match your system response. For me, 1 second wasn't enough, 2 seconds worked.

A Simple Example

Clone or Fork https://github.com/dmh2000/autoreload to a working directory

The example code is in autoreload/example. The example contains:

- main.go : web server that also starts autoreload
- views/*.html : web pages
- .air.toml : cosmtrek/air configuration file

Dependencies

Install the following dependencies

- golang
    - go install github.com/cosmtrek/air@latest

Web Pages

- add the 'reloader.js' script reference into the head section of each webpage

Run the example

```
$cd example
$go mod tidy
$air .
```

Test

- Make a change to the app source code. Either in example/main.go or example/views/{any page}
- Save the change
- On the terminal running 'air', you should see that Air restarted the Go app
- On the browser, you should see the changes pop up after a second or two