Webassembly

From WASM to WASI

Admin

- **Exam review? Preparation?**
  - Undergrad vs grad?
- **Project questions?**
- **Other?**

WebAssembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning

WebAssembly

- **So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning**
- **Once upon a time long long ago….**
  - The fine people at netscape (yes **long** ago when I was young and many of you were newborns) needed a scripting language to make the web more dynamic

WebAssembly

- **So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning**
- **Once upon a time long long ago….**
  - The fine people at netscape (yes **long** ago when I was young and many of you were newborns) needed a scripting language to make the web more dynamic
  - And there was javascript

WebAssembly

- **So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning**
- **Once upon a time long long ago….**
  - The fine people at netscape (yes **long** ago when I was young and many of you were newborns) needed a scripting language to make the web more dynamic
  - And there was javascript
  - And devs look on it and saw that it was bad
  - But it was the only choice

WebAssembly

- **So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning**
- **Once upon a time long long ago….**
  - The fine people at netscape (yes **long** ago when I was young and many of you were newborns) needed a scripting language to make the web more dynamic
  - And there was javascript
  - And devs look on it and saw that it was bad
  - But it was the only choice
  - So javascript became a big deal (And Node was born etc)

WebAssembly History

- **But but javascript**
  - What are some of its perceived limitations?
  - Lucky volunteer(TM)

WebAssembly History

- **But but javascript**
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
    - Lucky volunteer(TM)?

WebAssembly History

- **But but javascript**
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
      - For lots of stuff this doesn’t matter – latency for the Boston-Singapore database query is higher than the interpreted language slowdown.
    - If we haven’t covered it yet – when does it matter?

WebAssembly History

- **But but javascript**
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
      - For lots of stuff this doesn’t matter – latency for the Boston-Singapore database query is higher than the interpreted language slowdown.
    - If we haven’t covered it yet – when does it matter?
      - Games
      - Local computation-heavy work
      - Anything else?

WebAssembly History

- **So in 2013 – asm.js**
  - A strict subset of javascript
  - Designed to allow statically typed languages with manual memory management to be cross compiled to a subset of the javascript vm
    - So give me an example of the target language
    - Lucky volunteer(TM)?
  - Provided really good performance.

Webassembly History

- **In 2015 the powers that be decided to evolve asm.js into webassemby.**
- **By 2017 the MVP was out and implemented in “all major browsers”**
  - Firefox, chrome, safari, and IE (later also the chrome based edge)
  - A stack based virtual machine compile target
  - For manual memory management languages
  - Go has to compile the go runtime into the webassembly binary.

WebAssembly History

- **In 2025 WebAssembly 3.0 released.**
  - Headline features:
    - Garbage collector available in the stack machine itself.
    - 64 bit mode available
      - Prior to this 4GB max addressable memory
  - Coming soon/Feature flag protected features
    - Concurrency
    - ‘wide’ (128-bit) arithmetic.

Wasi

- **Wasi**
  - WebAssembly System Interface
  - Attempt to use webassembly outside of browser
    - In leaf nodes
    - How could this be helpful?

Wasi

- **Wasi**
  - WebAssembly System Interface
  - Attempt to use webassembly outside of browser
    - In leaf nodes
    - How could this be helpful?
    - If we have an obscure hardware/os combo that go doesn’t support
    - WASI might.
      - Envisioned to be used in IoT a lot.
      - Also includes automatic

WASM and WASI

- **Webassembly is downloaded from net and run automatically in your browser**
  - What could possibly go wrong?

WASM and WASI

- **Webassembly is downloaded from net and run automatically in your browser**
  - What could possibly go wrong?
  - Security issues? Like what?

WASM and WASI

- **Webassembly is downloaded from net and run automatically in your browser**
  - What could possibly go wrong?
  - Security issues? Like what?
  - Opens files on your computer – and does what?
  -

WASM and WASI

- **Webassembly is downloaded from net and run automatically in your browser**
  - What could possibly go wrong?
  - Security issues? Like what?
  - Opens files on your computer – and does what?
    - Fantastic question!
      - Uploads?
      - Encryption lock?
      - LLM training data?
      - Other?

WASM and WASI

- **Webassembly in browser locks out all local file access**
  - Get data via http
  - Or go:embed (more on this later in the semester)
  - Sandboxed runtime
- **WASI runs on local machine**
  - Can set very strict limits on what it is allowed to access.

Go and WebAssembly

- **Go was not the language Mozilla had in mind**
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?

Go and WebAssembly

- **Go was not the language Mozilla had in mind**
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?
  - Manual memory management
  - When go is compiled to webassembly so is its runtime
    - So application is minimum of 2mb
    - Hows that?
    - Lucky volunteer(TM)?

Go and WebAssembly

- **Go was not the language Mozilla had in mind**
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?
  - Manual memory management
  - When go is compiled to webassembly so is its runtime
    - So application is minimum of 2mb
    - Hows that?
    - Lucky volunteer(TM)?
    - Kinda crappy for a web commerce store – pretty ok for a 3d game

Go and Webassembly actually

- **Pure go translates to web assembly pretty easily.**
  - Go with foreign dependencies is harder (qt gtk etc)
- **In fact go has provided us with tools to make go apps web assembly even easier for a while**
  - `In your go install, there is a misc folder, and in there is a wasm_exec.html, we'll use this as your index.html`
- **Make a new folder for your running web project**
  - Copy <path to go>/misc/wasm/wasm_exec.html to your project
  - Replace test.wasm with your wasm file. – which we will generate momentarily

Go 1.21 (August 2023)

- **Go 1.21 increase support for wasi and wasm**
  - WebAssembly
    - The new go:wasmimport directive can now be used in Go programs to import functions from the WebAssembly host.
    - The Go scheduler now interacts much more efficiently with the JavaScript event loop, especially in applications that block frequently on asynchronous events.
  - WebAssembly System Interface
    - Go 1.21 adds an experimental port to the WebAssembly System Interface (WASI), Preview 1 (GOOS=wasip1, GOARCH=wasm).
    - As a result of the addition of the new GOOS value “wasip1”, Go files named *_wasip1.go will now be ignored by Go tools except when that GOOS value is being used. If you have existing filenames matching that pattern, you will need to rename them.

Now you need to web serve

- **At this point you could just run a heavy duty webserver like ngix**
  - And if we were deploying this to lots of people we would
  - But for development lets use a simple webserver that I pulled from a tutorial (reference link included)
- **`See next slide`**
  - `Put it into a single file, compile it to an executable`
  - `And then when we have our wasm ready we will run it`

The simplest web server

```
package main
//https://itnext.io/webassemply-with-golang-by-scratch-e05ec5230558
import (
  "log"
  "net/http"
)

const(
  AddSrv = ":8080"
  TemplatesDir = "."
)

func main() {
  log.Printf("listening on %q...", AddSrv)
  fileSrv := http.FileServer(http.Dir(TemplatesDir))
  if err := http.ListenAndServe(AddSrv, fileSrv); err!=nil{
    log.Fatal(err)
  }
}
```

Let’s Try

- **Lets try this now with the version of the project from the last topic**
  - The Fyne project
  -

Cross Compiling

- **What is ‘cross-compiling’?**

Cross Compiling

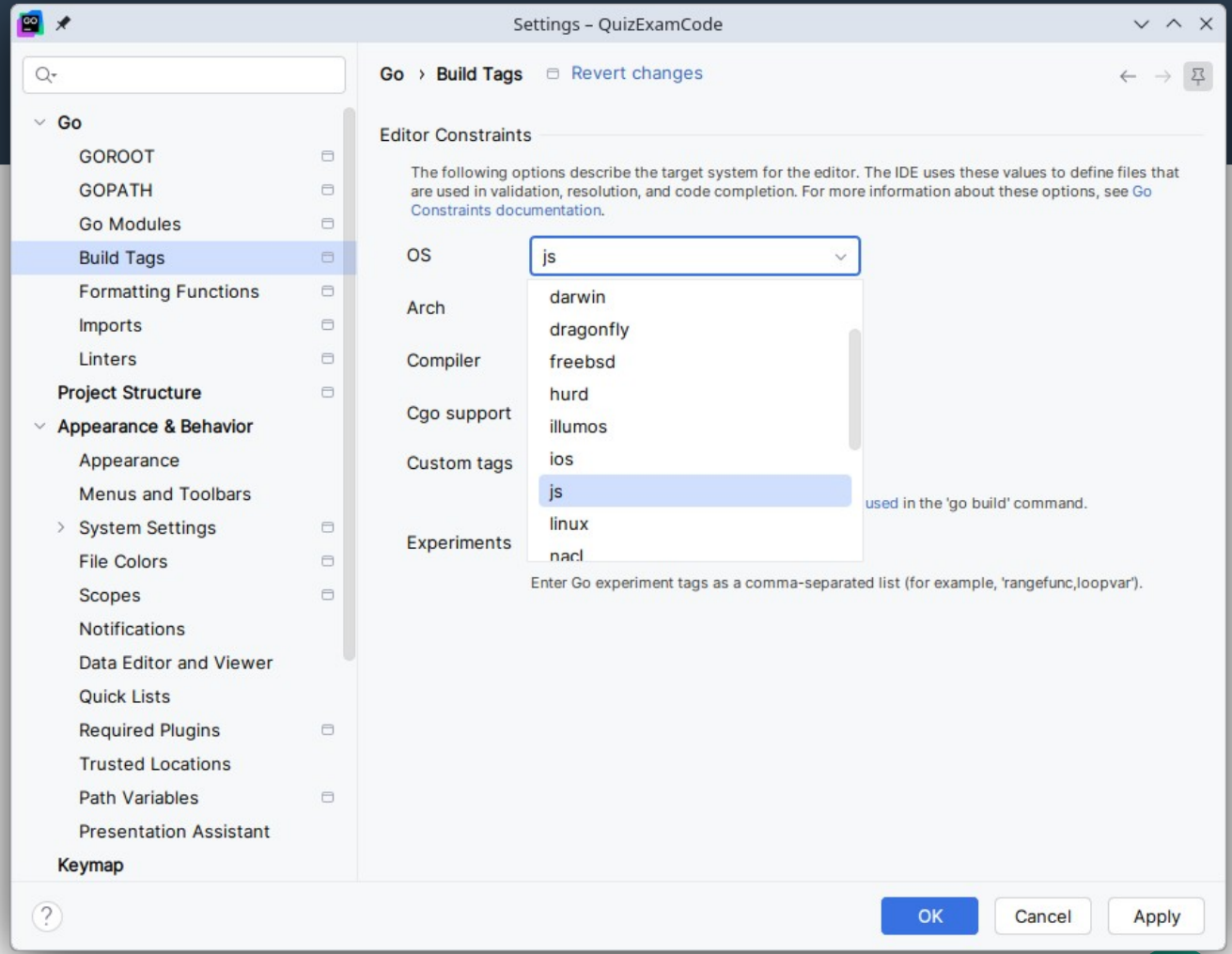
- **What is ‘cross-compiling’?**
  - Building an executable for one platform on a different platform
    - Eg: using windows on x86 architecture to build a program for a Mac on ARM architecture.
  - Webassembly (for now at least) always gets cross compiled.
    - Nobody running webassembly as their actual hardware architecture.
    -

Building From GoLand

- **You can do this painlessly in modern goland**
  - Open up the settings/preferences dialog
    - <file> <settings>Or <goland><Settings> in mac land
  - Open the <go> option and select the <build tags> option
  - In the OS option choose ‘js’
  - In the Arch option choose wasm
    - Now you are building for web assembly
    - And your executable will no longer run on your local system as before

1000 words

- **A picture is worth a thousand words**
- **So here**
  - Is the dialog in question

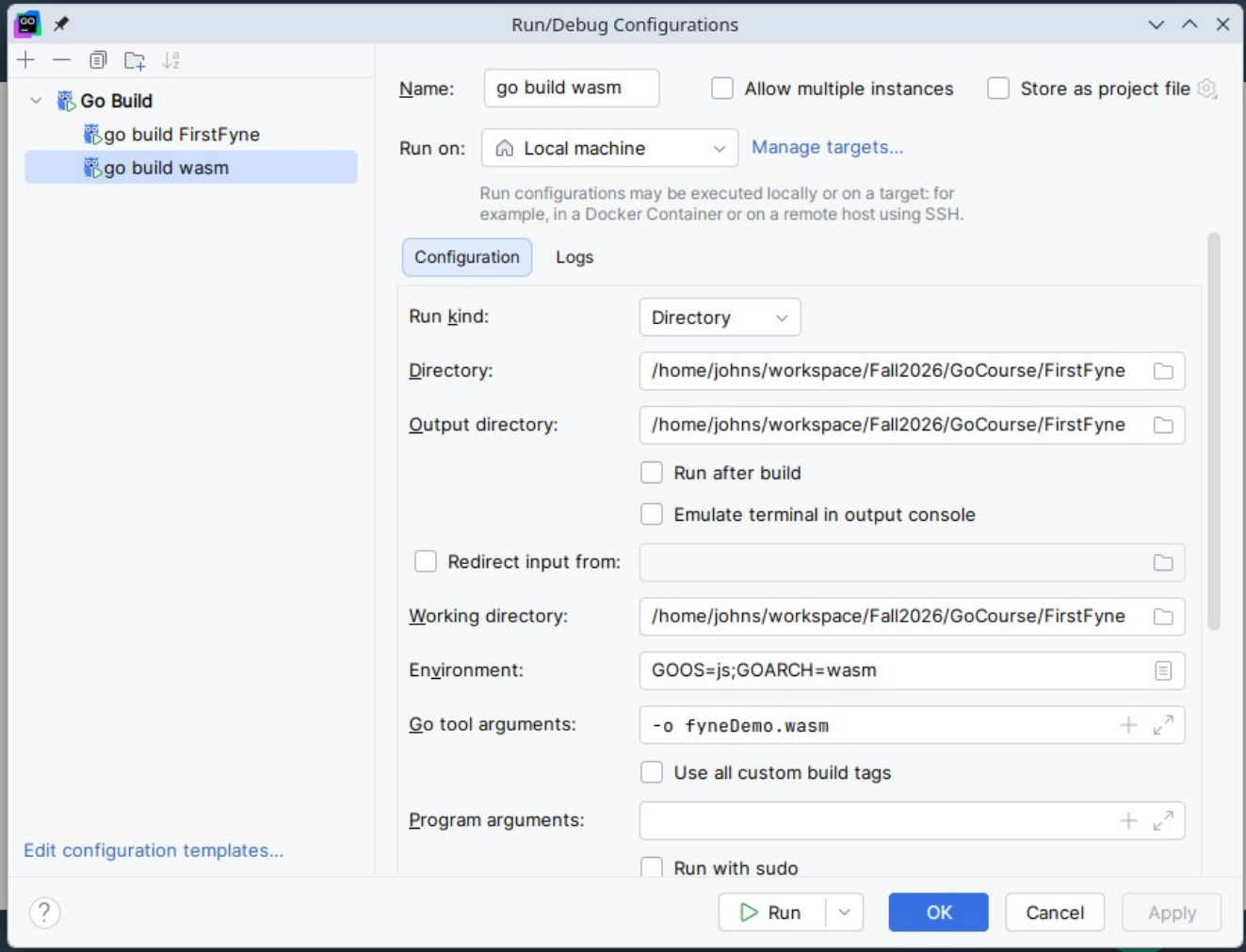


Edit Configurations

- **One last thing to do, go to <run> menu and <edit configurations>**
  - Hit the ‘+’ button to add a new configuration
  - Give it a new name (I like to put wasm in the name)
  - Make run kind *directory*
  - Uncheck ‘run after build’
  - Add two new environment variables GOOS=js;GOARCH=wasm
  - Change go tool arguments to -o <your program>.wasm
  - And hit ok
  - Then run your new configuration, nothing is run this time, but you should have a wasm file in your folder, move it to your web project folder

Another 1000 words

- Here is an image of one of my sample configs in goland



What about No Goland/Command Line

- **So if we are using goland, then web assembly is easy**
  - But go and webassembly without goland is still not bad.
  - At the command line
    - Set the GOOS and GOARCH environment variables
    - For example from the doc I put in the news:
    - GOOS=js GOARCH=wasm go build -o ../../assets/json.wasm <your file name>
  - But the command line is scary??!?!?
    - No but….

Let’s Run it

- **Run your compiled web server in the folder with your index.html and wasm program**
- **Then go to http://localhost:8080**
  - And lets see your program in the web browser.

Go:embed

- **Amazing added feature in go. (go 1.16 2021)**
  - You can embed files directly into the go program (the final executable) so all you have to give someone is a single executable file
  - Text files, images etc

To use go:embed

- **Usage**
  - You need the go:embed directive in a comment immediately over a global (or at least package wide) variable, which will hold the embedded asset
- **Eg:**
  - ```
    //go:embed assets/*
    var EmbeddedAssets embed.FS
    ```
- **This will take everything in the assets subfolder of the project and treat it as a file system**
    ```

Using the data from go:embed in a program

- **Here is an example function loading a file from the embedded directory**
  - I called it in the main function like this

```
data := loadEmbeddedTextFile("data/sampleData.json")
```

```
func loadEmbeddedTextFile(name string) []string {
	embeddedFile, err := EmbeddedAssets.ReadFile(name) //reads file into byte slice
	if err != nil {
		log.Fatal("Error:", err) //will kill program after logging error
	}
	return strings.Split(string(embeddedFile), "\n")
}
```

Project?

- **If we haven’t already, lets look at the next project.**

