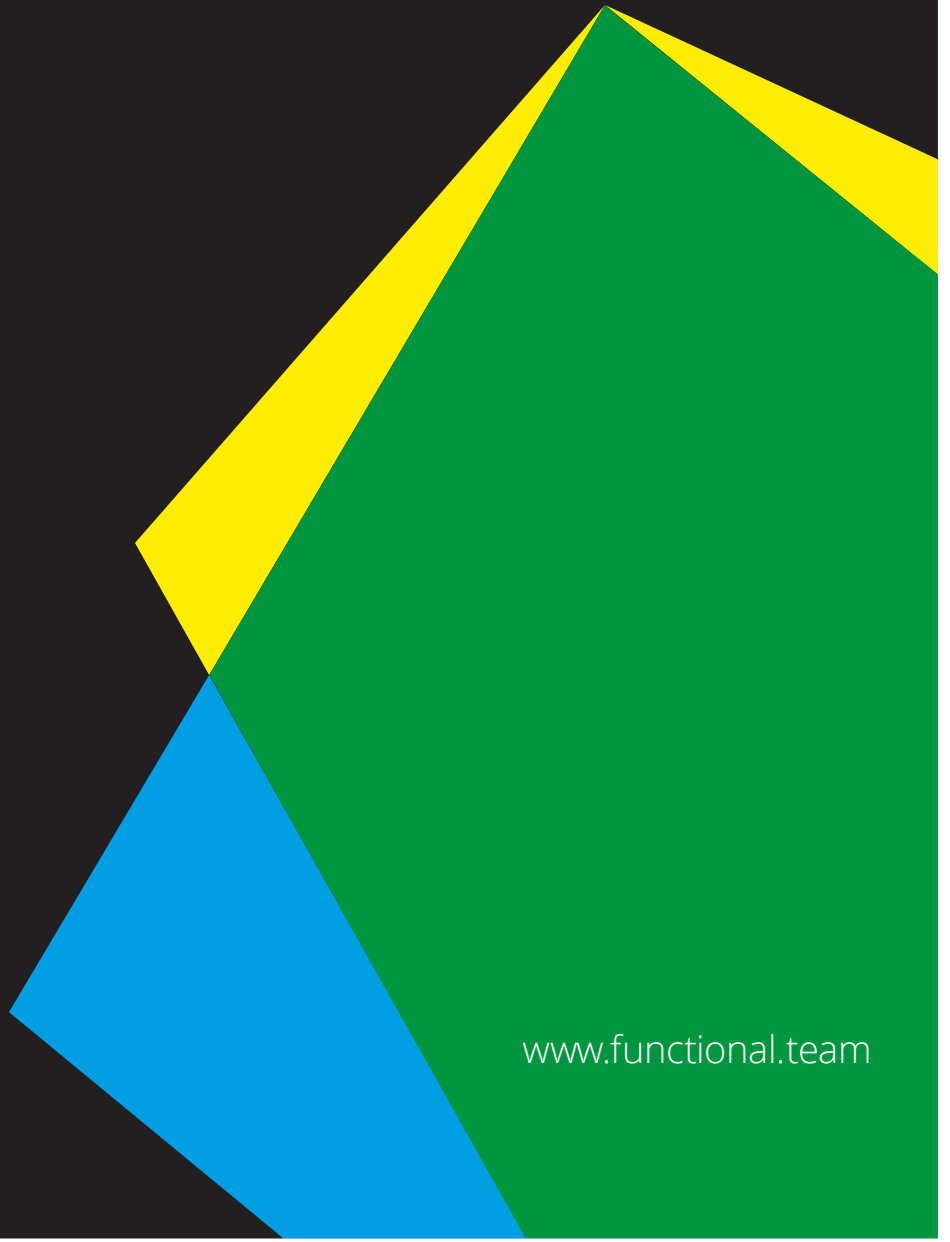
functional

Use Azure SDK for GO in WASM



www.functional.team

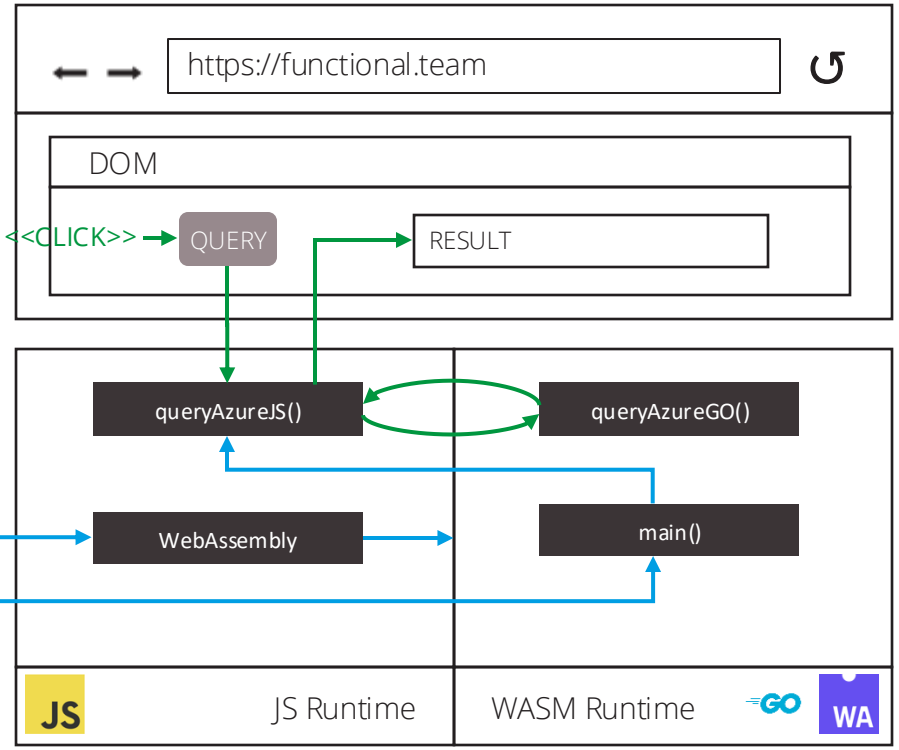
Scaffolding

```
<button onClick="queryAzureJS()">
  QUERY!
</button>

<script src="wasm_exec.js"></script>

<script>
  const go = new Go();

  WebAssembly.instantiateStreaming(
    fetch("main.wasm"),
    go.importObject)
  .then(
    (result) => {
      go.run(result.instance);
    }
  )
</script>
```




```
package main

import "syscall/js"

func queryAzureGo(
  this js.Value,
  args []js.Value) any {
    return "Results"
}

func main() {
  js.Global().Set(
    "queryAzureJS",
    js.FuncOf(queryAzureGo)
  )

  <-make(chan bool)
}
```

```
main.wasm
```

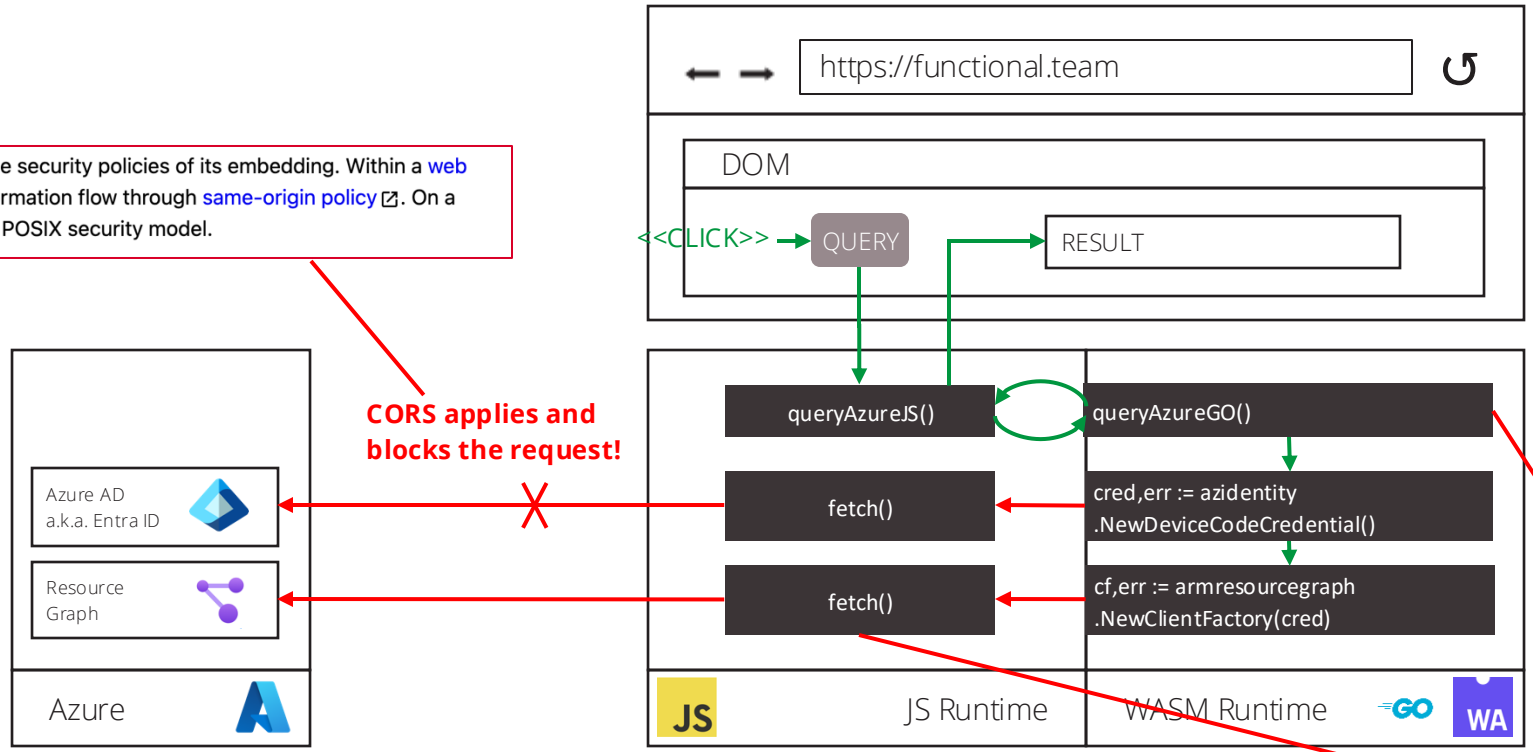
```
$ GOOS=js GOARCH=wasm go build -o main.wasm .
```

```
main.go
```

Issues

Additionally, each module is subject to the security policies of its embedding. Within a web browser, this includes restrictions on information flow through same-origin policy. On a non-web platform, this could include the POSIX security model.

https://webassembly.org/docs/security/



func FuncOf added in go1.12

```
func FuncOf(fn func(this Value, args []Value) any) Func
```

Invoking the wrapped Go function from JavaScript will pause the event loop and spawn a new goroutine. Other wrapped functions which are triggered during a call from Go to JavaScript get executed on the same goroutine.

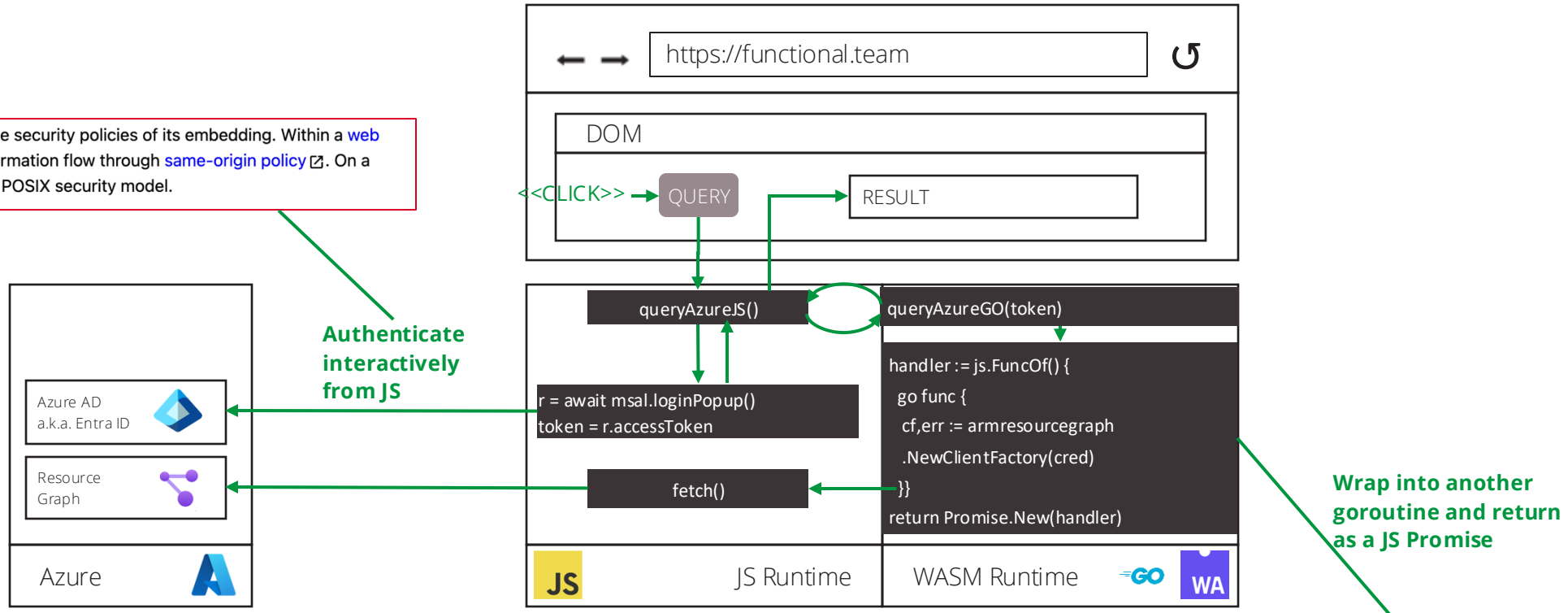
As a consequence, if one wrapped function blocks, JavaScript's event loop is blocked until that function returns. Hence, calling any async JavaScript API, which requires the event loop, like fetch (http.Client), will cause an immediate deadlock. Therefore a blocking function should explicitly start a new goroutine.

https://pkg.go.dev/syscall/js#FuncOf

Issues

Additionally, each module is subject to the security policies of its embedding. Within a web browser, this includes restrictions on information flow through same-origin policy. On a non-web platform, this could include the POSIX security model.

https://webassembly.org/docs/security/



func FuncOf added in go1.12

```
func FuncOf(fn func(this Value, args []Value) any) Func
```

Invoking the wrapped Go function from JavaScript will pause the event loop and spawn a new goroutine. Other wrapped functions which are triggered during a call from Go to JavaScript get executed on the same goroutine.

As a consequence, if one wrapped function blocks, JavaScript's event loop is blocked until that function returns. Hence, calling any async JavaScript API, which requires the event loop, like fetch (http.Client), will cause an immediate deadlock. Therefore a blocking function should explicitly start a new goroutine.

https://pkg.go.dev/syscall/js#FuncOf