Using Go in React Native

Daniel Wiese
dan@humon.io
CTO @ Humon
humon.io

Boston React Native Meetup
21-February-2019

humon

Introduction

Building native packages using Go and using them in your React Native app:

- Motivation
- Overview and Introduction to Go
- A motivating problem
- Building native packages using `gomobile` and using then in a React Native app
- Testing Go packages
- Other benefits of Go and React Native
- Summary

The complete minimum working example app and this presentation can be found:

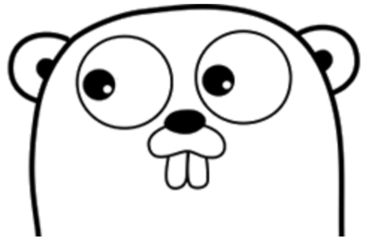
- https://github.com/dpwiese/react-native-gomobile-demo
- https://github.com/dpwiese/go-gomobile-demo
- http://danielwiese.com/using-go-in-react-native.pdf

Motivation

When and why to use native code, and why Go in your React Native app

- **Improve performance**
    - React Native has only a single JavaScript thread
    - Computationally intensive tasks can be time consuming and slow the UI
- **Obfuscate/hide code**
    - JavaScript code in a React Native app can be grabbed relatively easily
    - Access to proprietary algorithms can be restricted to those who need it
- **Add one language to a project instead of two**
    - Speed up development and reduce cost
    - Reduce platform specific bugs when implementing on multiple platforms
- **And more**
    - Use existing Go libraries
    - Javascript is tied to UI – cannot run background tasks
    - Share code with Go based backend

Overview of Go



- Golang - compiled, statically typed language created at Google in 2007
- Syntactically similar to C, but more concise and readable
  - Type inference with combined declaration/initialization operator `x := 1`
  - No `do` or `while`, iteration with `range`
- Fast compilation
- The main Go distribution includes tools for building, testing, and analyzing code
  - Package management with `go get`
  - Testing with `go test`
  - Standardized style with `gofmt` and `golint`
- Why use Go
  - Go is built with concurrency in mind
  - Go is good for complex projects
- See a list of Go Users including: Google, Dropbox, Medium, Uber, and many more

Introduction to Go

A Tour of Go provides a very nice walkthrough to introduce the language. You can find more Go resources at the end of this presentation. The following will give a brief introduction to Go.

- Every Go program is made up of packages
- Programs start running in package main
- Consider the following code, `example.go`:

```go
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("My favorite number is", rand.Intn(10))
}
```

Introduction to Go

- This program can be compiled and run from the command line using `go run`:

  ```
  $ go run example.go
  ```

- You can read about Go naming conventions: effective_go.html#names
  - Packages

    > lower case, single-word names; no underscores or mixedCaps

  - Use camel case instead of snake case

    > use MixedCaps or mixedCaps rather than underscores to write multiword names

  - Note the following regarding visibility of names in Go: Exported Identifiers

    > Names are as important in Go as in any other language. They even have semantic effect: ***the visibility of a name outside a package is determined by whether its first character is upper case.***

Introduction to Go

- When importing a package, you can only refer to it's exported names
- Any unexported names are *not accessible* from outside the package
- Go does not use public/private keywords for visibility of an identifier

```go
package main

import (
  "fmt"
)

func willNotExport() {
  fmt.Println("Are you there?")
}

func WillExport() {
  fmt.Println("Here I am!")
}
```

You can read more about naming and organization: The Go Blog: Organizing Go Code

Introduction to Go

- A sample function implementation

```go
func add(x int, y int) int {
    return x + y
}
```

- Variables are declared with the type last:

```go
var i int = 1
```

- the `:=` short assignment statement can be used in place of a `var` declaration with implicit type

```go
i := 1
```

- Go basic types are introduced in A Tour of Go: Basics 11
  - `bool`, `string`, sized and signed integers, sized floats and complex numbers

Introduction to Go

- Go has pointers
- Go is pass by value: a function always gets a copy of the thing being passed
- Defer
  - Schedules a function call (the deferred function) to be run immediately before the function executing the `defer` returns
  - Useful, for example, to remember to close files
- Slices
  - An array has a fixed size
  - Slices wrap arrays to give a more general, powerful, and convenient interface to sequences of data
  - Slice are dynamically-sized, flexible view into the elements of an array, that is they hold references to an underlying array
  - In practice, slices are much more common than arrays

Prerequisites

- Go
- Gomobile
- Xcode with command line tools (for iOS, covered in this presentation)
- Android Studio, NDK, Java (for Android, not covered here)

So now we've been introduced to Go, and are hopeful about how it can help improve our React Native app - let's keep going!

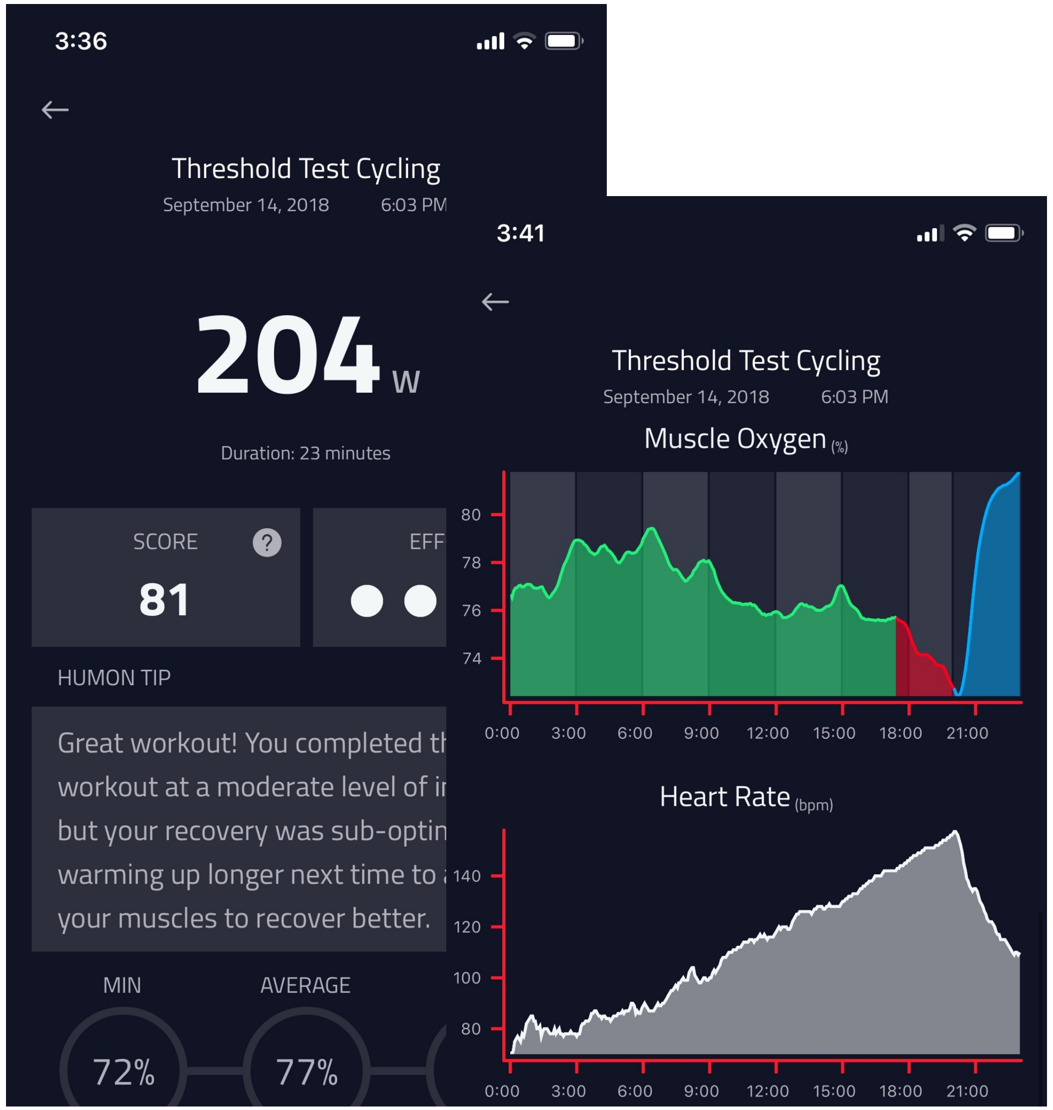
A Motivating Problem

- Let's say we want to make an app in React Native to solve a differential equation
- We'll take a numeric approach which requires many iterations to converge, making it very computationally intensive

While this is probably not the most realistic example, used here only for illustrative purposes, many more complex and realistic examples *do* exist, including:

- Processing audio
- Manipulating images
- Operations on large amounts of data
- Games
- Machine learning
- ...and many more

But this simple example will do...



Great workout! You completed th
workout at a moderate level of in
but your recovery was sub-optin
warming up longer next time to a
your muscles to recover better.

A Motivating Problem

Here's the first few lines of a JavaScript implementation of our differential equation solver:

```
export function solveBVP(callback: function) {
  // Some constants defining the problem
  const xMin = -1.0;
  const xMax = 1.0;
  const epsilon = 0.1;
  const maxIterations = 250000;
  const maxResidual = Math.pow(10, -11);

  // Run code here...

  callback({ theSolution });
}
```

- ***Running the example above takes on the order of about 10 seconds!***
- This is running on the iOS simulator, and much too slow for just about anything
- *Let's see if we can solve this problem a little quicker with Go*

Gomobile

Gomobile is a tool for building and running mobile apps written in Go. See the following links:

- Godoc: Command gomobile
- Golang Mobile Github Wiki

Types of applications:

- Native
- SDK

Gomobile uses `gobind`, which generates language bindings that make it possible to call Go functions from Java and Objective-C. Read more: Command gobind.

- Basically generate bindings from a Go package that can be invoked from Java (through C and the Java Native Interface (JNI)?) and Objective-C

Building a Go Package

- Here's a sample implementation of the same few lines of the function `SolveBVP` in Go
    - These few lines don't look too dissimilar from our JavaScript implementation from above...
- Name the package and function(s) recalling that only functions which begin with a capital letter are exported

```go
// Name the package
package sample

func SolveBVP() float64 {
  // Some constants defining the problem
  const xMin := -1.0
  const xMax := 1.0
  const epsilon := 0.1
  const maxIterations := 250000
  const maxResidual := 1.0e-11

  // Run code here...

  return theSolution
}
```

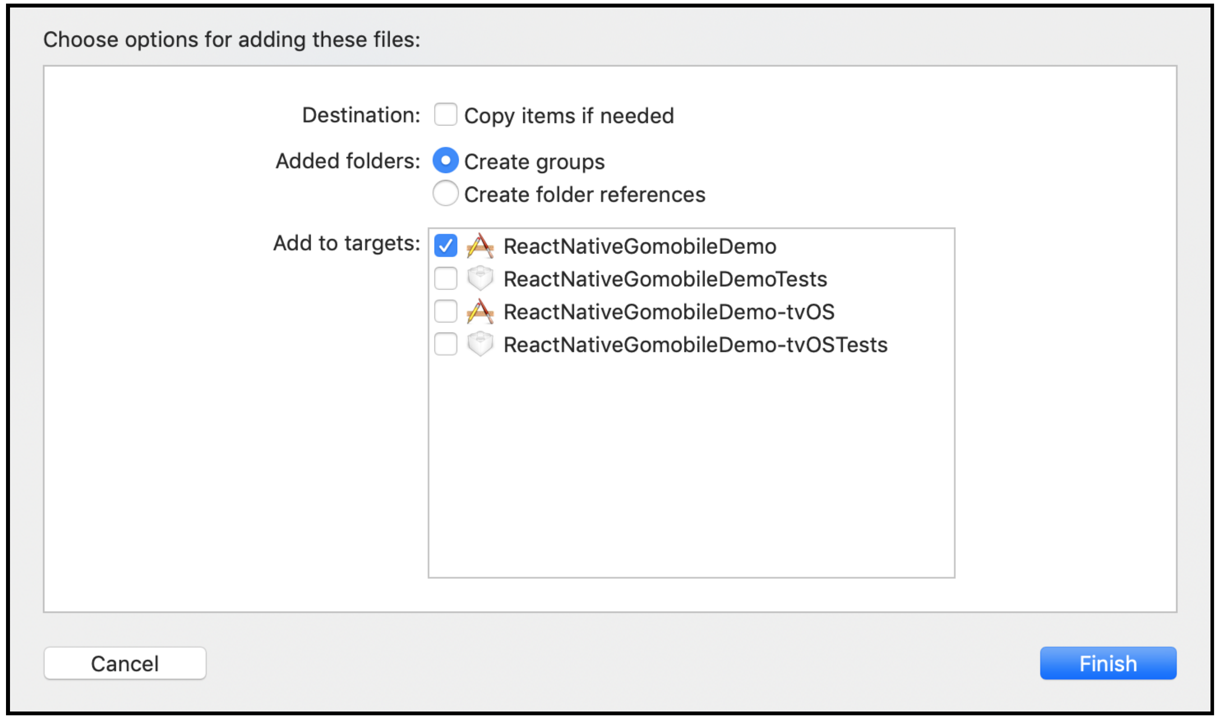
Building a Go Package

- Now we can generate our iOS Framework using the following command. To learn more about the build flags, use `go help build`.

```
$ gomobile bind -x -v -target=ios github.com/dpwiese/go-gomobile-demo
```

- This will generate `sample.Framework` that can now be included into your Xcode project.
- To generate `sample.aar` to use in an Android app, just set `-target=android`

This .Framework is ready to drag-and-drop into our Xcode project!



Choose options for adding these files:

Destination: Copy items if needed [ ]

Added folders: [✓ Create groups] Create folder references

Add to targets:
- [✓] ReactNativeGomobileDemo
- [ ] ReactNativeGomobileDemoTests
- [ ] ReactNativeGomobileDemo-tvOS
- [ ] ReactNativeGomobileDemo-tvOSTests

Using the Generated Framework

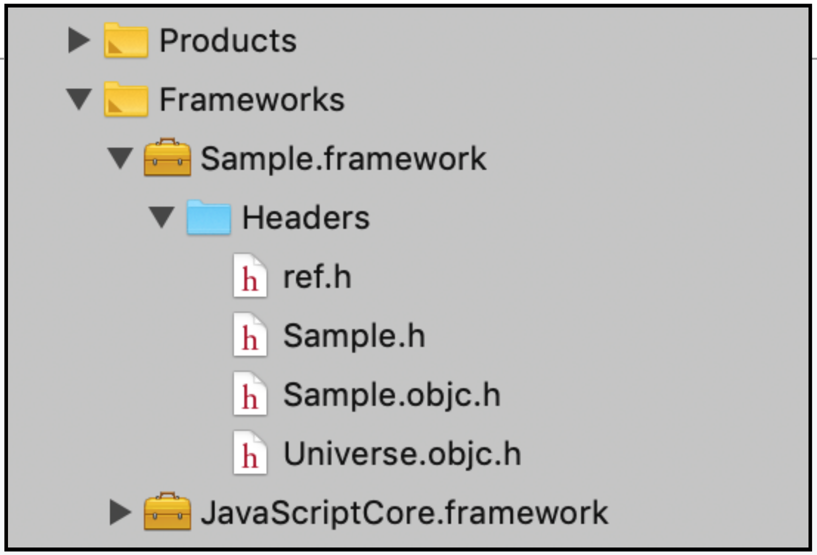
- To call `SolveBVP` from native code, will need to implement a bridge
- The name of the exposed Go function can be found in the header `Sample.objc.h` of the Framework that was automatically generated by gomobile:

```
#ifndef __Sample_H__
#define __Sample_H__

@import Foundation;
#include "Universe.objc.h"

FOUNDATION_EXPORT double SampleSolveBVP();

#endif
```



- The pattern should be clear, as: `<package_name>` `<function_name>`

Using the Generated Framework

- For the bridge, we'll create two files: a header and implementation
- Most of what is contained in these files is boilerplate, with the exception of the desired name of the interface (how we will access these functions from JavaScript) and the names and arguments of the functions themselves.

```
//  Computation.h
//  ReactNativeGoDemo
//
//  Created by Daniel Wiese on 1/28/2019.

#ifndef Computation_h
#define Computation_h

#import <React/RCTBridgeModule.h>
#import <Sample/Sample.h>

@interface Computation : NSObject <RCTBridgeModule>
@end

#endif
```

```objc
//  Computation.m
//  ReactNativeGoDemo
//
//  Created by Daniel Wiese on 1/28/2019.

#import <Foundation/Foundation.h>
#import "Computation.h"
#import <Sample/Sample.h>

@implementation Computation

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(solveBVP: (RCTResponseSenderBlock)callback)
{
  double elapsedTime = SampleSolveBVP();

  NSMutableArray *array = [[NSMutableArray alloc] initWithCapacity:0];
  [array addObject:[NSNumber numberWithDouble:elapsedTime]];

  callback(array);
}

@end
```

Using the Generated Framework

- In the implementation above that the function `solveBVP` is exported with a callback
- The Go code will be called across the bridge from the JavaScript side and the callback will fire when Go has finished
- The argument passed to the callback is an array. React Native Docs: Native Modules

> `RCTResponseSenderBlock` accepts only one argument - an array of parameters to pass to the JavaScript callback.

> The return type of bridge methods is always void. ***React Native bridge is asynchronous, so the only way to pass a result to JavaScript is by using callbacks or emitting events.***

> The name of the method exported to JavaScript is the native method's name up to the first colon.

Using the Generated Framework

Now we can import and call our native function:

```
import { solveBVP } from './computation';

Computation.solveBVP(solution => console.log(solution));
```

- With a single package written in Go and some very basic Objective-C, we've been able to implement the same function that we previously had in Javascript
- While the Android implementation was not shown, bridging with Java is very easy as well

  The Go implementation provides an order of magnitude performance benefit (computation time) over JavaScript, all while running on a separate thread.

- *However, the above example seems a bit contrived. If our function doesn't take any arguments, and always returns the same thing, we can just hardcode the answer in our JavaScript implementation!*

Passing Data Across the Bridge

- Let's pass some data across the bridge to our Go package
- The following basic Go types can be passed across the bridge: Type Restrictions
    - Signed integer and floating point
    - String and boolean
    - Byte slices
    - Functions, interfaces, and structs, whose parameters/methods are supported types
- Byte slices, JSON strings, and base64 strings are useful options. Let's say we want to pass some of the problem parameters across the bridge into Go
- Let's put these configuration parameters into a JSON object and pass it to our Go function

Unmarshalling JSON

- Because of Go's static typing, we need to use care to parse JSON
- Let's say we want to pass the following JSON configuration from our React Native app:

```
{
  "solverConfig": {
    "epsilon": 0.1,
    "maxIterations": 250000,
    "maxResidual": 1e-11,
    "domain": {
      "min": -1,
      "max": 1
    }
  }
}
```

- To parse this JSON in Go, we need to define the following types
- Recall again that for the name of the fields to be visible outside the package in which they are defined, the names all must begin with a capital letter

Unmarshalling JSON

```go
type configDatumType struct {
    SolverConfig solverConfigType `json:"solverConfig"`
}

type solverConfigType struct {
    Epsilon float64 `json:"epsilon"`
    MaxIterations int `json:"maxIterations"`
    MaxResidual float64 `json:"maxResidual"`
    Domain domainType `json:"domain"`
}

type domainType struct {
    Min float64 `json:"min"`
    Max float64 `json:"max"`
}
```

- Now, we can pass a JSON string from JavaScript, across the bridge and into Go, where it can be parsed and the data used within the function
- Any other fields present in JSON but not defined in the types will be ignored

Unmarshalling JSON

- The following Go snippet shows how, using the types defined above, the JSON string can be parsed and the values easily accessed for use in our Go function

```go
func SolveBVPWithInputs(configString string) float64 {

    // Unmarshal the JSON string configString
    var configDatum configDatumType
    json.Unmarshal([]byte(configString), &configDatum)

    // Access the unmarshalled configuration
    xMin := configDatum.SolverConfig.Domain.Min
    xMax := configDatum.SolverConfig.Domain.Max
    epsilon := configDatum.SolverConfig.Epsilon
    maxIterations := configDatum.SolverConfig.MaxIterations
    maxResidual := configDatum.SolverConfig.MaxResidual

    // Run code here...

    return theSolution
}
```

Base64 Strings

- Another useful datatype for passing across the bridge is base64 string
- The following code provides some basic functions in Go and JavaScript for converting between numeric types (e.g. a slice of float64, array of numbers) and a base64 string
- Such functions have been useful when passing data between JavaScript and Go

Base64 Strings: Go

```go
import (
  b64 "encoding/base64"
  "bytes"
  "encoding/binary"
)
```

```go
func float64FromByteSlice(byteSlice []byte) float64 {
  return math.Float64frombits(binary.LittleEndian.Uint64(byteSlice))
}

// Convert base64 string to slice of float64
func float64SliceFromBase64String(str string) []float64 {
  byteSlice, _ := b64.StdEncoding.DecodeString(str)
  var floatSlice []float64

  for i := 0; i < len(byteSlice)/8; i++ {
    floatSlice = append(floatSlice, float64FromByteSlice(byteSlice[8*i:8*(i+1)]))
  }

  return floatSlice
}
```

Base64 Strings: Go

```go
// Convert slice of float64 to base64 string
func base64StringFromFloat64Slice(arr []float64) string {
  buf := new(bytes.Buffer)
  binary.Write(buf, binary.LittleEndian, arr)
  data := buf.Bytes()

  return b64.StdEncoding.EncodeToString([]byte(data))
}
```

- The above sample functions allow us to convert between base64 strings and float64 in Go
- Modifying these functions to accommodate other data types should be straighforward
- Handing of data in this way might remind you of Typed Arrays in JavaScript, where buffers and views are used

Base64 Strings: JavaScript

```
import {decode as atob, encode as btoa} from 'base-64'

// Convert base64 string to number array
export function base64StringToNumberArray(str: string): Array<number> {
  var uint8_array = Uint8Array.from(atob(str), c => c.charCodeAt(0));
  var buffer = new ArrayBuffer(uint8_array.byteLength);
  new Uint8Array(buffer).set(uint8_array);
  return new Float64Array(buffer);
}

// Convert number array to base64 string
export function numberArrayToBase64String(arr: Array<number>): string {
  var buffer = new ArrayBuffer(arr.length*8);
  new Float64Array(buffer).set(arr);
  return btoa(String.fromCharCode.apply(null, new Uint8Array(buffer)));
}
```

Adding Flexibility to Types

- For example, say we have an array of x-y points in our React Native app, where each element is an object and the y value may represent a string, number, etc.

```
{ x: number, y: any }
```

- You can do this using an empty interface.

```
type value struct {
  X float64 `json:"x"`
  Y interfaceType `json:"y"`
}

type interfaceType interface {

}
```

- This, while it circumvents all type checking and can be misused has been quite useful at times in practice.

Updating our Native Bridge

- Now that we've seen how to work with some data passed from JavaScript to Go, we need to update our bridge
- The changes are pretty simple...
- In JavaScript, we'll define our configuration object, and pass it across the bridge with `JSON.stringify()`
- In Go, we'll return the solution, instead of just a number, as a JSON string containing additional information

```
//
//  Computation.m
//  ReactNativeGoDemo
//
//  Created by Daniel Wiese on 1/28/2019.
//

#import <Foundation/Foundation.h>
#import "Computation.h"
#import <Sample/Sample.h>

@implementation Computation

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(solveBVPWithInputs: (NSString*)configString
                            callback: (RCTResponseSenderBlock)callback)
{
  NSString *string = SampleSolveBVPWithInputs(configString);

  NSArray *stringArray = @[string];
  callback(stringArray);
}

@end
```

Updating our Native Bridge

- And now in our callback we can parse the JSON output and handle the values
- Examples with other data types are available in the example repositories

Checkpoint

- With that, we should have the ability to:
  - Write packages in Go
  - Build them into native modules
  - Pass data back-and-forth from React Native

Testing a Go Package

- Use the go testing framework. Go package testing
- When writing tests using Go Testing, don't have to expose functions via export (or in this case first capital letter) in order to test them, or use a module like rewire.
- So say we want to test our Go function in file `sample.go`:

```go
package sample

func ConcatenateStrings(string1 string, string2 string) string {
  return string1 + string2
}
```

> Package testing provides support for automated testing of Go packages. Use it with the "go test" command, which automates execution of any function of the form

```go
func TestXxx(*testing.T)
```

where `Xxx` does not start with a lowercase letter.

Testing a Go Package

- Create a test file `sample_test.go`
- Define package name and imports

```go
package sample

import (
	"reflect"
	"testing"
)

// Body to follow...
```

Testing a Go Package

```go
func TestConcatenateStrings(t *testing.T) {
  type testCases struct {
    string1  string
    string2  string
    expected string
  }

  var tests = []testCases{
    {string1: "hello", string2: "world", expected: "helloworld"},
    {string1: "foo", string2: "bar", expected: "foobar"},
  }

  for ic, tc := range tests {
    var result = ConcatenateStrings(tc.string1, tc.string2)
    if !reflect.DeepEqual(result, tc.expected) {
      t.Errorf("[Test #%d] Expected is: %s, got: %s", ic+1, tc.expected, result)
    }
  }
}
```

Testing a Go Package

- To run this test, use the following command:

```
$ go test -v
```

- Your output will look something like the following:

```
=== RUN   TestConcatenateStrings
--- PASS: TestConcatenateStrings (0.00s)
=== RUN   TestAddNumbers
--- PASS: TestAddNumbers (0.00s)
...
```

- Or if a failure:

```
=== RUN   TestConcatenateStrings
--- FAIL: TestConcatenateStrings (0.00s)
sample_test.go:34: [Test #2] Expected is: foo bar, got: foobar
...
```

- Go testing package also offers benchmarking

Linting and Formatting

- You can get help formatting your Go code with Gofmt

  ```
  $ gofmt -d .
  ```

  where the `-d` flag shows a diff, instead of rewriting files.

- Golint can be helpful as well:

  ```
  $ golint <filename>.go
  ```

Other Benefits: Code Obfuscation

- Often proprietary code may be used in a client application, and it is often beneficial to offload computation to the client to reduce backend costs
- With a React Native app, the JavaScript bundle is distributed, unencrypted, with the app
- After archiving an app in Xcode, for example, you'll find within it as shown below, the JavaScript bundle:

```
~/Library/Developer/Xcode/Archives/2019-02-13/ReactNativeGomobileDemo 2-13-19,
10.05 AM.xcarchive/Products/Applications/ReactNativeGomobileDemo/main.jsbundle
```

- Although the code has been minified, it is available for all to see:

Other Benefits: Code Obfuscation

```
__d(function(g,r,i,a,m,e,d){var n=r(d[0]);Object.defineProperty(e,"__esModule",{
value:!0}),e.solveBVPWithoutInputs=function(n){t("{\"solverConfig\":{\"epsilon\"
: 0.1, \"maxIterations\": 250000, \"maxResidual\": 1.0e-11, \"domain\": {\"min\"
: -1.0, \"max\": 1.0 }}}",n)},e.solveBVPWithInputs=t;var o=n(r(d[1]));function t
(n,t){var f=JSON.parse(n),s=f.solverConfig.domain.min,u=f.solverConfig.domain.ma
x,l=f.solverConfig.epsilon,p=f.solverConfig.maxIterations,v=f.solverConfig.maxRe
sidual,w=(u-s)/100,h=1,y=0,A=new Array(101),M=new Array(101),c=new Array(100),x=
new Array,C=new Array(101),I=new Array(101),_=new Array(100),P=new Array(101),b=
(new Date).getTime();for(i=0;i<=101;i++)A[i]=1;for(;h>v&&y<p;){for(i=1;i<101;i++
)C[i]=-A[i]/(2*w)-l/Math.pow(w,2),I[i]=2*l/Math.pow(w,2),_[i]=A[i]/(2*w)-l/Math.
pow(w,2),P[i]=0;for(C[0]=0,I[0]=1,_[0]=0,P[0]=1,C[100]=0,I[100]=1,P[100]=-1,i=1;
i<=101;i++)I[i]=I[i]-_[i-1]*C[i]/I[i-1],P[i]=P[i]-P[i-1]*C[i]/I[i-1];for(M[100]=
P[100]/I[100],i=99;i>=-1;i--)M[i]=(P[i]-_[i]*M[i+1])/I[i];for(y+=1,c[0]=0,i=1;i<
100;i++)c[i]=M[i]*((M[i+1]-M[i-1])/(2*w))-l*((M[i-1]-2*M[i]+M[i+1])/Math.pow(w,2
));var B=M.map(function(n,o){return 1.99*(n-A[o])});A=A.map(function(n,o){return
n+B[o]});var D=c.map(function(n,o){return Math.abs(n)});h=Math.max.apply(Math,(0
,o.default)(D)),x.push(h)}t({iteration:y,error:h,time:(new Date).getTime()-b})}}
,347,[1,15]);
```

Other Benefits: Code Obfuscation

- Reverse engineering native code is more difficult
- This extra level of difficulty sufficient
- Similar accessibility of JavaScript bundle with Expo?

Other Benefits

Separation of Concerns

- Increases engineering efficiency
  - Engineers responsible for designing, implementing, and testing complex algorithms need not have much of an understanding of mobile development, nor do they need their environment set up for mobile development
  - They can focus on writing Go, designing a clean interface, testing, and ultimately provide a "black box" to the mobile team, with well-defined inputs and ouputs
  - Likewise, the mobile team need not concern themselves with the inner workings of such a "black box", and with a clearly defined interface, can quickly and easily write or modify the bridge and JavaScript code to leverage the Go Framework when provided.
- Made easy by the above approach

"Function parity" across Android and iOS

Summary

- Go can be used to create native packages for use in Android and iOS
- Some basic examples were provided showing the syntax of a simple Go function
- The examples also showed how to access the Go functions from JavaScript via a native bridge
- Some examples were provided and discussed to deal with passing different data types back-and-forth between JavaScript and Go
- Some tools that accompany the main Go distribution were introduced
- Benefits of native code as well as Go specifically were discussed

Resources

- **Go language specification:** a reference manual for the Go programming language
  - https://golang.org/ref/spec
- **A Tour of Go:** interactive examples introducing Go
  - https://tour.golang.org/
- **How To Write Go Code:** The go programming language docs
  - https://golang.org/doc/code.html
- **Effective Go:** A large resource on how to writing clear, idiomatic Go code
  - https://golang.org/doc/effective_go.html
- **Go By Example:** A long list of concise Go examples
  - https://gobyexample.com/
- **The Go Playground**
  - https://play.golang.org/
- **The Go Blog**
  - https://blog.golang.org/

Resources (continued)

- **Go package sources**
  - https://golang.org/src/
- **Go Code Review Comments:** common comments made during reviews of Go code
  - https://github.com/golang/go/wiki/CodeReviewComments
- **React Native: Why and How to Build Your Native Code in Go:** first blog post I stumbled upon when starting this work
  - https://hackernoon.com/react-native-why-and-how-to-build-your-native-code-in-go-9fee492f0daa
- A curated list of awesome Go frameworks, libraries and software.
  - https://github.com/avelino/awesome-go
- A short blog post about writing unit tests using testing package
  - https://blog.alexellis.io/golang-writing-unit-tests/
- Go on Wikipedia
  - https://en.wikipedia.org/wiki/Go_(programming_language)

Questions?