Go and GUIs

Showing your stuff to users

Admin

- We’ll do a midterm that I’ll give you in a couple/few of weeks.
- Now lets talk about cup-o-go podcase Episode that I asked you all to listen to for today.
- Next week Monday schedule on Feb 21

GUIs

- What kind of GUI programming have you done.
  - Quick survey

GUIs

- My guess before talking with you all
  - Java: JavaFX, maybe android and swing,
  - Kotlin: android?
  - Swift: iOS native?
  - Python: pyQT or similar?
  - javascript: react, angular etc
  - Anything else?

How does it all work?

- Arranging components on the window?
  - 1980ish-2010ish?

How does it all work?

- Arraigning components on the window?
  - 1980ish-2010ish?
    - Put controls on window, use some sort of layout to arrange them. Use static properties to paint them the right way.
  - 2010ish-now

How does it all work?

- Arraigning components on the window?
  - 1980ish-2010ish?
    - Put controls on window, use some sort of layout to arrange them. Use static properties to paint them the right way.
  - 2010ish-now
    - Use some sort of layout to arrange them, use CSS to make them look nice.

Threads and event handling

- How do most GUI libraries handle threads and events?

Threads and event handling

- How do most GUI libraries handle threads and events?
  - Main thread spawns new thread which will run the GUI, your main thread becomes vestigial
    - java's main 'method' or
    - pythons first method called in the `if __name__ == '__main__'` block
  - How do events get handled?

Threads and event handling

- How do most GUI libraries handle threads and events?
  - Main thread spawns new thread which will run the GUI, your main thread becomes vestigial
    - java's main 'method' or
    - pythons first method called in the `if __name__ == '__main__'` block
  - How do events get handled?
    - The GUI library has an event loop, you provide callback functions in your code.
    - Event loop? Callbacks? (let the students answer)

New Approaches to GUIs

- In the last few years we've seen a bunch of new approaches to GUIs taking hold
  - Lots of CSS and javascript, but more.
  - Awesome go:
    - fyne - Cross platform native GUIs designed for Go based on Material Design. Supports: Linux, macOS, Windows, BSD, iOS and Android.
    - go-astilectron - Build cross platform GUI apps with GO and HTML/JS/CSS (powered by Electron).
    - go-gtk - Go bindings for GTK.
    - go-sciter - Go bindings for Sciter: the Embeddable HTML/CSS/script engine for modern desktop UI development. Cross platform.
    - gotk3 - Go bindings for GTK3.
    - gowd - Rapid and simple desktop UI development with GO, HTML, CSS and NW.js. Cross platform.
    - qt - Qt binding for Go (support for Windows / macOS / Linux / Android / iOS / Sailfish OS / Raspberry Pi).
    - ui - Platform-native GUI library for Go. Cross platform.
    - Wails - Mac, Windows, Linux desktop apps with HTML UI using built-in OS HTML renderer.
    - walk - Windows application library kit for Go.
    - webview - Cross-platform webview window with simple two-way JavaScript bindings (Windows / macOS / Linux).

Go GUI libraries

- Lets take a look
  - Some old standbys
    - Go bindings for GTK
    - Go bindings for QT
  - Some 2010s javascript CSS stuff
    - Lots of electron based stuff (someone tell us about electron)
      - go-astilectron
      - Wails
      - go-sciter

New language new approaches

- But go is new – and lots of traction for wholly new approaches to GUIs
  - Two GUI libraries for go that have the most buzz in the last 6 months:
    - Fyne (Materials based design)
    - Gio (creator is Skandinavian so pronounce gi-oh)
      - Immediate mode gui

Retained Mode GUI

- In Retained mode GUI (most common for the last 30-40 years)
  - Graphics library ‘retains’ (holds onto and controls) the visuals
    - Often holds a copy of the data you are using
  - Client calls (that is your code) just say to graphics library “make these changes and update please”

Immediate Mode GUI

- The graphics library gives you the client the primitives
  - And client (you) tell it what and when to render to the screen
  - Big change: if you are determining when to render what else do you have to be in charge of?

Immediate Mode GUI

- The graphics library gives you the client the primitives
  - And client (you) tell it what and when to render to the screen
  - Big change: if you are determining when to render what else do you have to be in charge of?
    - Event loop – window resize needs to re-render, so you need to catch and handle those events.
  - Often done in go by having functions that return other functions as return values
    - Was challenging fun to learn with grad students – think we'll do retained mode here.

What is really going on

- Computer is just 1s and 0s so what is really going on deep inside the computer for these GUIs?
- How do we see interface components on the screen?

What is really going on

- Computer is just 1s and 0s so what is really going on deep inside the computer for these GUIs?
- How do we see interface components on the screen?
  - We

Go UI Libraries

- See https://github.com/avelino/awesome-go#gui
- Wrappers around C++ libraries
  - go-gtk, gotk3, therecipe/qt, nuklear and nucular
- Go + html + css
  - Wails, go-astilectron, webview, gowd, go-sciter
- Pure go
  - Fyne, gio (immediate mode)
- And one last:
  - Ebitenui (a UI library aimed at both general UI and game UIs)

Fyne

- I'll use Fyne in this class for GUI
  - 'materials design' GUI
    - Originally a google standard
    - Designed to have the same application look the same across platforms.
  - For last year, can now compile to webassembly
    - And deploy applications to web.
  - Otherwise classic GUI development feel

Hello Fyne

- Here is the official hello world app

```
package main

import (
	"fyne.io/fyne/v2/app"
	"fyne.io/fyne/v2/widget"
)

func main() {
	a := app.New()
	w := a.NewWindow("Hello World")

	w.SetContent(widget.NewLabel("Hello World!"))
	w.ShowAndRun()
}
```

- Other than the terrible variable names – does it make sense?

Now let's extend our program

- Now lets extend our University program from last time.
  - So we ask the user for the University keyword in a window
  - And then display the data in a list.

Get Universities

```
import (
	"encoding/json"
	"fmt"
	"io/ioutil"
	"net/http"
	"os"
)
func GetUniversityData(searchTerm string) *[]UniversityResponse {
	apiURL := fmt.Sprintf("http://universities.hipolabs.com/search?name=%s", searchTerm)
	response, err := http.Get(apiURL)
	if err != nil {
	fmt.Println("Error getting internet response.....\nCowardly quitting.....\n")
	os.Exit(-1)
	}
	defer response.Body.Close()
	bodyData, err := ioutil.ReadAll(response.Body)
	if err != nil {
	fmt.Println("Error reading response body")
	return nil
	}
	universities := make([]UniversityResponse, 2)
	if err = json.Unmarshal(bodyData, &universities); err != nil {
	fmt.Println("Error - could not translate json data to struct properly")
	}
	return &universities
}
```

Get Data is now a function that takes the search term and returns a slice

UniversityResponse is unchanged

Window Struct and List helpers

```
type DisplayWindow struct {
	UniversitiesData *[]UniversityResponse
	DataDisplay      *widget.List
	UniversityInput  *widget.Entry
}

func getUniversityData() {
	searchTerm := window.UniversityInput.Text
	window.UniversitiesData = GetUniversityData(searchTerm)
}
```

This is the window struct we will use.

The functions below is called by the button when it is pressed.

List creation helpers

```
func GetDataLen() int {
    if window.UniversitiesData == nil {
        return 0
    }
    return len(*window.UniversitiesData)
}

func CreateListItem() fyne.CanvasObject {
    return widget.NewLabel("Universities will appear here")
}

func UpdateListItem(itemNum widget.ListItemID,
listItem fyne.CanvasObject) {
    UnivName := (*window.UniversitiesData)[itemNum].Name
    listItem.(*widget.Label).SetText(UnivName)
}
```

These three functions are needed by the fyne List factory function

The first one needs to tell how many items are in the list

The second one needs to tell what an empty list item will be

The third one need to tell how the empty item should be updated to show data

If the list should do something when selected, try making buttons instead of labels

Finally Main.go

```
import (
	"fyne.io/fyne/v2"
	"fyne.io/fyne/v2/app"
	"fyne.io/fyne/v2/container"
	"fyne.io/fyne/v2/widget"
)

var window DisplayWindow

func main() {
	displayApp := app.New()
	mainWindow := displayApp.NewWindow("University Data in a List")
	window = DisplayWindow{}
	window.UniversityInput = widget.NewEntry()
	window.UniversityInput.SetPlaceHolder("Enter University Name to search for...")
	getDataButton := widget.NewButton("Get University Data", getUniversityData)
	window.DataDisplay = widget.NewList(GetDataLen, CreateListItem, UpdateListItem)
	topPane := container.NewVBox(window.UniversityInput, getDataButton)
	contentPane := container.NewVSplit(topPane, window.DataDisplay)
	mainWindow.SetContent(contentPane)
	mainWindow.Resize(fyne.NewSize(400, 900))
	mainWindow.ShowAndRun()
}
```

Let's try it.

Project 2.1 coming tonight
Summarize here