Building modern desktop apps in Go

Serge Zaitsev / zserge.com / @zsergo
Mobile? Wearable? Desktop!

For getting job done with high performance, security and privacy.
WIMP Era

Window, Icon, Menu, Pointer. Ruling the world since 1973.
The Golden Age of WIMP UI
Post-WIMP Era (still evolving)
Well-designed and attractive typography makes content meaningful. Animation makes important parts stand out and helps with micro-interactions. Onboarding, common for web and mobile apps, helps more than traditional "F1" button. Visual trends may change and it should not be hard to update the UI without rewriting it all.
Electron Is Bad.
What do we want?

Common UI API for all platforms because we are lazy. Support for existing UI frameworks and UI kits (material, flat, etc) because we often can't design UI. Painless development, debugging, packaging and distribution because we have more important things to do.
When programmers make UI

- System information
- Board bar: each board is a set of columns
- User logo

- The fixed attribute determines how the column content moves to left/right
- Thumbs are displayed when the "Thumb" is pressed
- Files are sorted by type: directories, links, programs, documents

- *YOU MUST READ the "license term"; consider you have a license (register)
- Drag up or down to change the height of the viewer

- Information about active partition, directory, file, link target
- If you want to sort your projects and notes by time, change the sort mode for each column

- Toolbar
- The hint bar shows hints for controls over which the mouse is moved
- Set the number of columns, for each board
- Quickly change to a new set of colors
Option 1.

Reuse some modern browser that is already installed.

Option 2.

Reuse the browser engine that comes with the OS (and every desktop OS now comes with a decent browser engine).
Lorca

github.com/zserge/lorca
Lorca

Chrome DevTools Protocol
Bind Go functions to JS
Call JS from Go
Control native window

API

ui, _ := lorca.New(...)
ui.Load(url)
ui.Bind("someFunc", func() {})
five := ui.Eval("2+3").Int()
<-ui.Done()
Example

```go
code
package main

import (
    "log"
    "net/http"
    "net/url"
    "github.com/zserge/lorca"
)

const html = `<!DOCTYPE html>
<html>
<head><title>Check URL</title></head>
<body>
  <input id="url" />
  <button id="check" onclick="checkURL(document.querySelector('#url').value)">Check</button>
  <div id="status"></div>
</body>
</html>

func main() {
    if err != nil {
        log.Fatal(err)
    }
    ui.Bind("checkURL", func(url string) {
        res, err := http.Get(url)
        if err == nil && res.StatusCode == http.StatusOK {
            ui.Eval("document.querySelector('#status').innerText = 'Online';")
        } else {
            ui.Eval("document.querySelector('#status').innerText = 'Offline';")
        }
    })
    defer ui.Close()
}

Example
```
Lorca!

Minimal (1KLOC). One dependency (websocket lib)
Simple API.
ES6 and modern CSS without Babel.
Decent debugger.

Lorca?

Window can't have fixed size.
Window global menu can't be controlled.
It still behaves more like a browser rather than an app.
WebView
github.com/zserge/webview
GTK+  
GtkWebKit2  

WinAPI  
MSHTML (OLE)  

Cocoa  
WKWebView  

WinAPI  
EdgeHTML (winrt)
Window

Set title.
Set size.
Optional: other window flags (border, full-screen mode, transparency)
Minimize/maximize/restore ...
(BYOF - bring your own features)

Browser

Load arbitrary URL (including data URIs).
Initialize with JS code when new page is loaded (before DOM is ready).
Evaluate JS code any time later.
Call native callback with a string argument from JS.
Go

ui, _ := webview.New()
ui.SetTitle("Hello")
ui.Load(url)
ui.Bind("foo", func(s string) {})
defer ui.Close()
ui.Run()

C++

webview w;
w.set_title("Hello");
w.load(url);
w.bind("foo", [](string arg){})
w.run()
Lorca

Disk: ~10MB
RAM: ~80MB
CPU: 2%

WebView

Disk: ~10MB
RAM: ~6MB
CPU: ~1.3%

Note: benchmarks are useless, just try it yourself.
Questions?

github.com/zserge/lorca
github.com/zserge/webview