POINTER

2025-04

Building Desktop Apps with Go

General
Use case and requirements
Choosing a framework
Examples

Go and GUI

Status:

- no built-in GUI library
- requires 3rd-party package/framework
- diverse approaches

Examples:

- Fyne, https://fyne.io/
- Gio, https://gioui.org/
- Wails, https://wails.io/

To watch:

- Cogent Core, https://www.cogentcore.org/

Fyne

Package:

- https://fyne.io/
- Apache License 2.0

Features:

- cross-platform: Windows, macOS, Linux, iOS, Android
- designed for ease of use
- rich set of built-in UI components
- layout system: VBox, HBox, Grid, Form, ...
- event handling system to manage user interactions: clicks, key presses, ...
- supports theming
- built-in support for internationalization (i18n)
- accessibility: screen reader, keyboard navigation, ...

Fyne (2)

```
package main
import (
    "fyne.io/fyne/v2/app"
    ...
)

func main() {
    // Create a new app
    myApp := app.New()

    // Create a new window
    myWindow := myApp.NewWindow("Hello Fyne")
    // Create a simple label
    helloLabel := widget.NewLabel("Hello, World!")
    // Add the label to the window
    myWindow.SetContent(container.NewVBox(
        helloLabel,
        widget.NewButton("Quit", func() {
            myApp.Quit()
        }),
    ))

    // Show the window and run the app
    myWindow.ShowAndRun()
}
```


GIO

Package:

- https://gioui.org/
- MIT License

Features:

- cross-platform: Windows, macOS, Linux, iOS, Android
- declarative approach to UI design
- rich set of built-in widgets
- custom widgets
- layout system: linear, grid, custom, ...
- supports styling and theming

GIO (2)

```
package main
import (
    "gioui.org/app"
    "gioui.org/io/system"
     ...
)

func main() {
    go func() {
        w := app.NewWindow()
        var ops op
        for {
            switch e := w.NextEvent().(type) {
            case system.DestroyEvent:
                return
            case system.FrameEvent:
                gtx := layout.NewContext(&ops, e)
                material.H4(theme, "Hello, Gio!").Layout(gtx)
                e.Frame(gtx.Ops)
            }
        }
    }()
    app.Main()
}

type op struct{}
func (op) Add(o *op) {}
var theme = material.NewTheme()
```


Wails

Package:

- https://wails.io/
- MIT License

Features:

- cross-platform: Windows, macOS, Linux
- separates backend (Go) from frontend UI (HTML, CSS, and JavaScript)
- selection of web technology for frontend: Vue, React, Svelte, ...
- two-way communication between Go backend and JavaScript frontend (ie, call Go functions from JavaScript, ...)
- supports hot reloading for backend and frontend
- built-in developer tools: debug, optimize, browser developer tools, ...
- allows customization of app window: title, size, icon, minimize, maximize, ...
- provides APIs to access file system
- supports native dialogs: file selection, save file, display message, ...

Wails (2)

```
package main
import (
    "github.com/wailsapp/wails/v2"
...
)

// App struct
type App struct{}
// NewApp creates a new App application struct
func NewApp() *App {
    return &App{}
}
// Greet returns a greeting for the given name
func (a *App) Greet(name string) string {
    return "Hello, " + name + "!"
}

func main() {
    // Create an instance of the app structure
    app := NewApp()
    // Create application with options
    err := wails.Run(&application.AppConfig{
        Bind: []interface{}{
            app,
        },
    })
    if err <> nil {
        println("Error:", err)
    }
}
```


Wails (3)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Wails App</title>
</head>
<body>
    <h1>Wails App</h1>
    <input type="text" id="nameInput" placeholder="Enter your name">
    <button onclick="greet()">Greet</button>
    <p id="greeting"></p>

    <script type="module">
        import { Greet } from '../wailsjs/go/main/App.js';

        async function greet() {
            const name = document.getElementById('nameInput').value;
            const greeting = await Greet(name);
            document.getElementById('greeting').innerText = greeting;
        }
    </script>
</body>
</html>
```


POINTER

Use case and requirements

Use case

POS application.

- touch-screen operation
- small scale display: 14″, 15″, 15.6″, ...
- landscape orientation
- retail-specific functions

Kiosk application:

- touch-screen operation
- large scale display: 27″, 32″, ...
- portrait orientation
- retail-specific functions

Requirements

Intuitive UI

- clean design with reduced distractions
- consistency through uniform design language for application
- easy navigation
- touch-friendly: buttons, icons, ‘tappable’, ...
- responsive layout: varying screen size, orientation, ...
- user-friendly messaging and feedback
- accessibility features

Multi-language support:

- multiple locales: language, currency, date/time, data formatting, ...

Branding:

- themes
- implementation of store’s branding

Requirements (2)

Checkout process:

- reduced number of steps to complete sale
- logical grouping of content: categories, products, payments, ...
- multiple payment options
- summary before sale

Information:

- visual browsing
- product navigation by category
- product details
- product search

Choosing a framework

Q&A

Logic and UI:

- within Go system, or ...
- separate backend (Go) and frontend (HTML, CSS, JavaScript)

UI design:

- built-in UI components, or ...
- flexibility with frontend framework (CSS, JavaScript)

Rich frontend:

- built-in UI components and layouts, or ...
- (re)use of framework with large system of libraries, tools, ...

Development skills:

- learn framework-specific UI components and layouts, or ...
- (re)use web development technologies

Q&A (2)

OS integration:

- framework-specific dialogs, or ...
- API to access native dialogs

Community support:

- Go-centric, or ...
- Go with web development

Use case:

- ‘standard’ UI: utility tool, configuration app, limited UI customization, or ...
- ‘customized’ UI: dashboard, data visualization, touch-screen app

Why did we select Wails?

Features:

- flexible UI design with ‘standard’ frontend framework
- implementation of rich frontend based on comprehensive set of pre-designed components, layouts, and tools
- intuitive and visually appealing user interface based on Material Design guidelines
- (re)use known web development technologies
- ‘customized’ UI for touch-screen apps

Selection of technology:

- Go
- Wails, https://wails.io/
- Vue, https://vuejs.org/
- Vuetify, https://vuetifyjs.com/

Why did we select Wails? (2)

Concerns:

- complexity and learning curve: requires knowledge of both Go and web technologies (HTML, CSS, JavaScript)
- dependency on web technologies: browser compatibility, web standards, ...
- performance: overhead when using web-based frontend, latency in communication between Go backend and JavaScript frontend, ...
- development tools, debugging: separate tools for each part of application, ...
- OS support: Android ?

Examples

Sample projects

Example 1:

- simple project
- Wails+Vanilla, Wails+Vue, Wails+Vuetify

Example 2:

- rich frontend
- Wails, Vue, Vuetify

References

Go software:

- https://go.dev/
- https://pkg.go.dev/

Articles:

- https://golangweekly.com/issues/*
- https://changelog.com/gotime/271 (Podcast)

Lists, projects:

- https://github.com/go-graphics/go-gui-projects
- https://github.com/benjamin-thomas/wails-elm-template?tab=readme-ov-file#note-on-performancealternatives

Thank you!

Contacts:

Branko Zečević, branko.zecevic@pointer.hr
Lovro Zečević, lovro.zecevic@pointer.hr
Fran Cvok, fran.cvok@pointer.hr