Andy Williams

fyne conf 2022

Best practices for excellent Fyne apps

Clean and maintainable code for the life of your project

Project Setup

fyne conf 2022

- Source Control
- Go module unique name
- Gofmt/goimports & go vet
  - golang.org/x/tools/cmd/goimports
- Static Check
  - honnef.co/go/tools/cmd/staticcheck
- Complexity check
  - github.com/fzipp/gocyclo/cmd/gocyclo

```
module github.com/andydotxyz/myapp

go 1.18

require (
        fyne.io/fyne/v2 v2.2.3
)
```

Automation of Checks

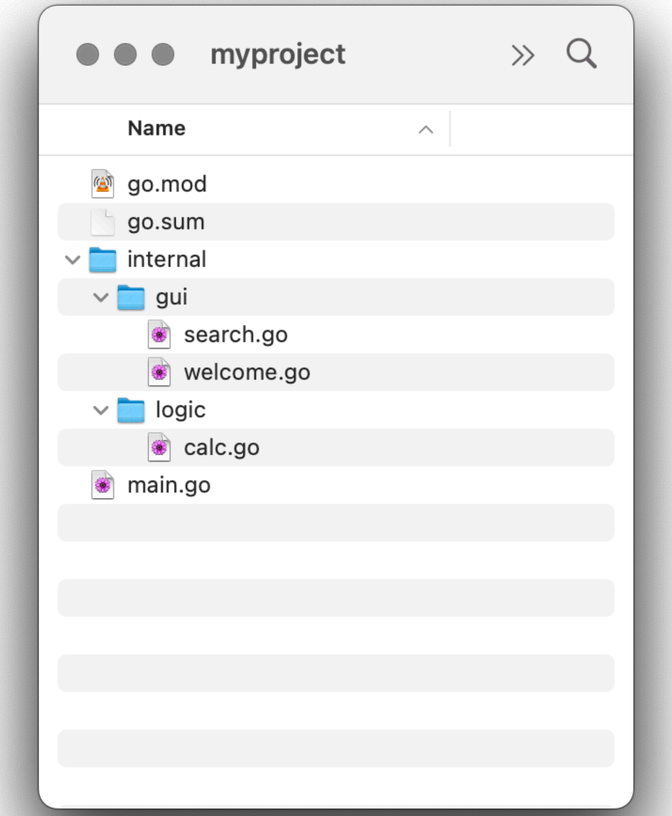
fyne conf 2022

- Git Hook (pre-commit)
  - Configured for each git clone (local)
- Continuous Integration
  - Run vet, formatter, static check
  - Execute tests
  - Check code coverage

```
#!/bin/sh
fail_if_err () {
        eval $2
        if [ "$?" -ne 0 ]; then
                echo "FAILED " $1
                exit 1
        fi
}

fail_if_err "FORMAT" "[ -z $(goimports -l .) ]"
fail_if_err "TEST"   "go test ./... > /dev/null"
fail_if_err "VET"    "go vet ./..."
fail_if_err "LINT"   "golint -set_exit_status \$(go list ./...)"
fail_if_err "CYCLO"  "gocyclo -over 30 ."
```

Packages

- Don’t add packages too soon
- `main` package at project root
- Internal packages for related code
- Separate UI from app logic
- One file for each app GUI area



Small files, small functions, clean code

fyne conf 2022

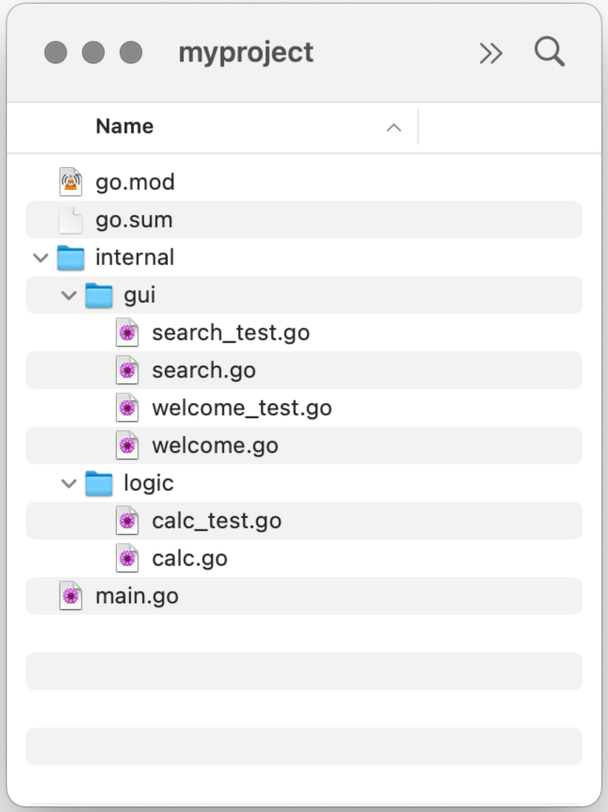
CLEAN CODERS

- Multiple files with clear names
- One main type per file
- Keep functions simple, one main purpose
- Split UI setup into many smaller functions

Unit testing

fyne conf 2022

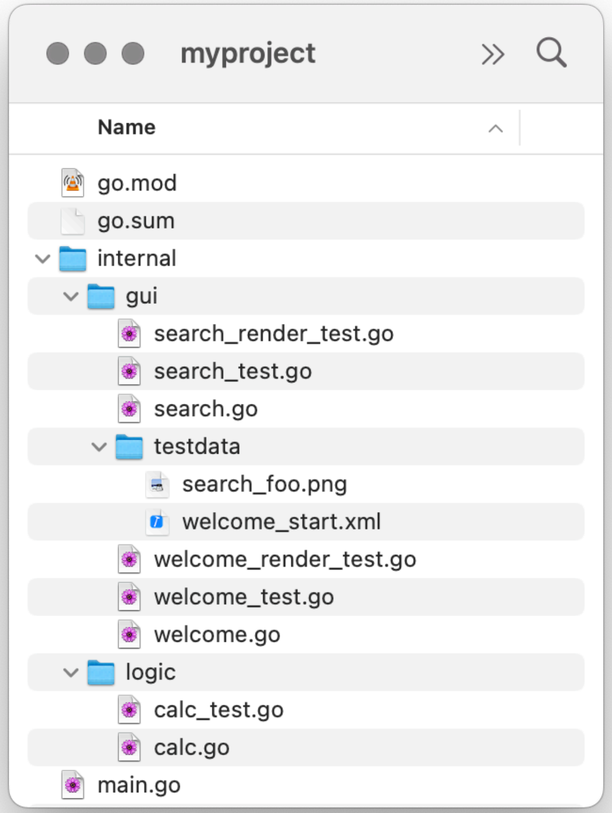
```go
package test

import (
    "testing"

    "github.com/stretchr/testify/assert"
    "fyne.io/fyne/v2/test"
    "fyne.io/fyne/v2/widget"
)

func TestText_Selected(t *testing.T) {
    e := widget.NewEntry()
    test.Type(e, "Hello")
    assert.Equal(t, "Hello", e.Text)

    test.DoubleTap(e)
    assert.Equal(t, "Hello", e.SelectedText())
    assert.Equal(t, 5, e.CursorColumn)
}
```



Render testing

fyne conf 2022

- Validate that a component or screen renders as expected
- Compare to .PNG or .XML golden file
  - `test.AssertImageMatches` compares visually
  - `test.AssertRendersToMarkup` compares structure
- `testdata` folder for comparison files
- `testdata/failed` will be used when tests fail



Use FyneApp.toml

fyne conf 2022

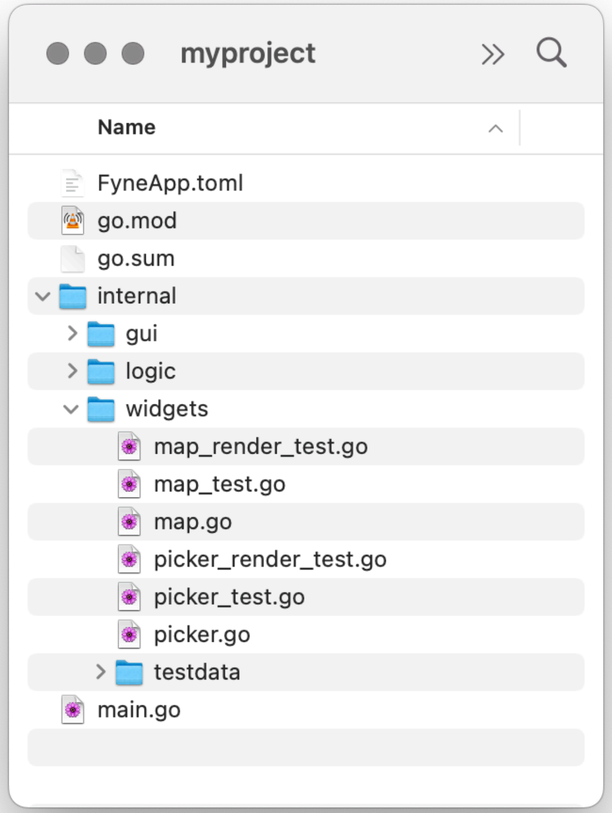
- Store metadata in repo
- Reproducible builds
- Avoiding build scripts
- Smoother release process

```
Website = "https://fyne.io"

[Details]
Icon = "img/fyne.png"
Name = "Fyne Demo"
ID = "io.fyne.demo"
Version = "1.0.0"
Build = 5
```

Custom Widgets

fyne conf 2022

- Maximum one per file
- Unit tests for all behaviour
- UI test for validating look/layout
- Expose behaviour only
- Store user state in widget
- Don’t store renderer reference



Custom Widgets - Behaviour API

- Create type that extends `BaseWidget`
- Call `ExtendBaseWidget()` in constructor
- Export fields that are used in configuration
- Create public methods that expose behaviour

Custom Widgets - Behaviour API

fyne conf 2022

```go
// Map is our custom widget
type Map struct {
    widget.BaseWidget

    Watermark  string
    zoom, x, y int
}

// NewMap creates a new instance of the map widget.
func NewMap() *Map {
    m := &Map{}
    m.ExtendBaseWidget(m)
    return m
}

// ZoomIn steps the zoom in one level.
func (m *Map) ZoomIn() {
    // zoom code
    m.Refresh()
}
```

Custom Widgets - Renderer

- Important state in widget
- Disposable information in renderer
- Read all state in `Refresh()`
- Adapt to size available in `Layout()`
- Free any resources in `Destroy()`

Custom Widgets - Renderer

fyne conf 2022

```go
func (m *Map) CreateRenderer() fyne.WidgetRenderer {
    return &mapRenderer{widget: m, cache: &canvas.Raster{}}
}

var _ fyne.WidgetRenderer = (*mapRenderer)(nil)

type mapRenderer struct {
    widget *Map
    cache  *canvas.Raster
}

func (m *mapRenderer) Destroy() {
}

func (m *mapRenderer) Layout(s fyne.Size) {
    m.cache.Resize(s)
}

func (m *mapRenderer) MinSize() fyne.Size {
    return fyne.NewSize(50, 50)
}

func (m *mapRenderer) Objects() []fyne.CanvasObject {
    return []fyne.CanvasObject{m.cache}
}

func (m *mapRenderer) Refresh() {
    // TODO draw the current map and apply m.widget.Watermark
}
```

Custom Widgets - SimpleRenderer

fyne conf 2022

- Helper for trivial widgets
- Avoid creation of custom renderer
- Pass object(s) to `NewSimpleRenderer`
- Send changes to objects in `Refresh()`
- Object will fill space available

Custom Widgets - SimpleRenderer

fyne conf 2022

```go
func (m *Map) CreateRenderer() fyne.WidgetRenderer {
    if m.cache == nil {
        m.cache = &canvas.Raster{}
    }
    return widget.NewSimpleRenderer(m.cache)
}
```

And Remember...

fyne conf 2022

- Export only what should be public
- Document all of your APIs
- Naming is important, make sure your API is clear

- Start simple, ask community for help