Go Package Project Structure

Reusable Go module with clean public API, internal implementation, and proper versioning.

#go #golang #package #library #module

PNG PDF Copy Prompt

Project Directory

```
mypackage/
    go.mod  Module path def…
    go.sum
    LICENSE
    README.md
    .gitignore
    doc.go  Package documen…
    mypackage.go  Main public API
    options.go  Functional opti…
    errors.go  Exported error …
    mypackage_test.go  Public API tests
    example_test.go  Runnable exampl…
  internal/  Private impleme…
    parser/
      parser.go
      parser_test.go
    utils/
      utils.go
  testdata/  Test fixtures
    sample.json
```

Why This Structure?

Go packages are simple by design. Public API lives at the root, private code in `internal/`. The module path in `go.mod` is your import path. Version with git tags (v1.0.0) for `go get` compatibility.

Key Directories

`mypackage.go` - Main public types and functions

`doc.go` - Package-level documentation for godoc

`internal/` - Implementation details, not importable

`example_test.go` - Runnable examples shown in docs

Functional Options Pattern

```
// mypackage.go
package mypackage

// Client is the main entry point for the package.
type Client struct {
    baseURL string
    timeout time.Duration
}

// New creates a Client with functional options.
func New(opts ...Option) *Client {
    c := &Client{timeout: 30 * time.Second}
    for _, opt := range opts {
        opt(c)
    }
    return c
}
```

Getting Started

1. `go mod init github.com/user/mypackage`
2. Create public API in root `.go` files
3. Add examples in `example_test.go`
4. `go test ./...`
5. `git tag v1.0.0 && git push --tags`

When To Use This

- Building a reusable library
- Code shared across multiple projects
- Publishing to pkg.go.dev
- Internal company packages
- SDK for an API

Trade-offs

`Flat structure` - Large packages may need subdirectories

`No cmd/` - Add cmd/ if you also need a CLI

`Versioning` - v2+ requires /v2 in module path

Best Practices

- Keep public API small and focused
- Use functional options for configuration
- Write examples that appear in godoc
- Use internal/ liberally for implementation
- Document exported symbols

Naming Conventions

`Package name` - Short, lowercase, no underscores

`Exported symbols` - PascalCase (Client, New, Option)

`Interfaces` - Single method: -er suffix (Reader, Writer)