Go Gin REST API Project Structure

Gin framework REST API. Fast HTTP performance, middleware support, and request binding.

#go #golang #gin #rest #api #http #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    main.go  Entry point
    go.mod
    go.sum
    Makefile
    .env.example
    .gitignore
  cmd/
    api/
      main.go
  internal/
    handler/  HTTP handlers
      handler.go  Base handler
      users.go
      auth.go
    middleware/
      auth.go  JWT middleware
      cors.go
      logger.go
    model/
      user.go
      request.go  Request DTOs
      response.go  Response DTOs
    service/
      user_service.go
      auth_service.go
    repository/
      user_repo.go
    router/
      router.go  Route definitio…
    config/
      config.go
  pkg/
    response/
      json.go
    validator/
      validator.go  Custom validato…
```

Why This Structure?

Gin is the fastest Go web framework with a martini-like API. It has built-in request binding, validation, and middleware support. This structure organizes handlers, middleware, and services for a production REST API.

Key Directories

`internal/handler/` - Gin handlers with request binding

`internal/middleware/` - Auth, CORS, logging middleware

`internal/model/` - Request/response DTOs with binding tags

`internal/router/` - Route groups and middleware chains

Gin Router Setup

```
// internal/router/router.go
func Setup(h *handler.Handler, mw *middleware.Middleware)
    r := gin.Default()

    r.Use(mw.CORS())

    api := r.Group("/api/v1")
    {
        api.POST("/login", h.Login)

        auth := api.Group("/", mw.Auth())
        {
            auth.GET("/users", h.ListUsers)
            auth.GET("/users/:id", h.GetUser)
        }
    }
    return r
}
```

Getting Started

1. `go mod init myproject`
2. `go get github.com/gin-gonic/gin`
3. `cp .env.example .env`
4. `go run cmd/api/main.go`

When To Use This

- Need high HTTP performance
- Want built-in request validation
- Building REST APIs with authentication
- Prefer established framework with large community
- Need middleware ecosystem (gin-contrib)

Trade-offs

`Less idiomatic` - Gin patterns differ from stdlib net/http

`Context wrapping` - gin.Context vs context.Context can confuse

`Dependency size` - Larger than Chi or stdlib

Naming Conventions

`Handlers` - Methods on Handler struct (h.GetUser)

`Middleware` - Return gin.HandlerFunc

`Request DTOs` - Suffix with Request (CreateUserRequest)

`Route groups` - api, auth, admin prefixes