Go Chi REST API Project Structure

Chi router with middleware, request binding, and structured RESTful endpoints.

#go #golang #chi #rest #api #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    main.go  Entry point
    go.mod
    go.sum
    .env  Environment var…
    .env.example
    .gitignore
    Makefile  Build and run c…
  cmd/  Application ent…
    api/
      main.go  HTTP server
  internal/  Private applica…
    config/
      config.go  Env parsing wit…
    handler/  HTTP handlers
      handler.go  Base handler wi…
      users.go
      health.go
    router/
      router.go  Chi router setup
      middleware.go  Custom middlewa…
    model/
      user.go
      errors.go  API error types
    repository/  Data access lay…
      user_repo.go
    service/  Business logic
      user_service.go
  pkg/  Shared utilities
    response/
      json.go  JSON response h…
```

Why This Structure?

Chi is the most popular lightweight Go router. It composes well with standard `net/http` middleware, supports URL parameters, and stays close to idiomatic Go patterns. This structure uses `internal/` for private code and separates handlers, services, and repositories.

Key Directories

`cmd/api/` - Application entrypoint, starts HTTP server

`internal/handler/` - HTTP handlers with dependency injection

`internal/service/` - Business logic, independent of HTTP

`internal/repository/` - Database operations and queries

`pkg/` - Shared code that could be used externally

Chi Router Setup

```go
// internal/router/router.go
func NewRouter(h *handler.Handler) chi.Router {
    r := chi.NewRouter()

    r.Use(middleware.Logger)
    r.Use(middleware.Recoverer)
    r.Use(middleware.RequestID)

    r.Get("/health", h.Health)

    r.Route("/api/v1", func(r chi.Router) {
        r.Get("/users", h.ListUsers)
        r.Get("/users/{id}", h.GetUser)
        r.Post("/users", h.CreateUser)
    })

    return r
}
```

Getting Started

1. `go mod init myproject`
2. `go get github.com/go-chi/chi/v5`
3. `cp .env.example .env`
4. `go run cmd/api/main.go`

When To Use This

- Building REST APIs with multiple endpoints
- Need middleware for logging, auth, CORS
- Projects with 5+ routes
- Team projects needing clear structure
- Want to stay close to stdlib patterns

Trade-offs

`External dependency` - Chi is well-maintained but adds a dependency

`More boilerplate` - Handler/service/repo layers add files

`Learning curve` - internal/ package visibility rules take time to learn

Naming Conventions

`Packages` - lowercase, singular (handler, model, service)

`Files` - Lowercase with underscores (user_repo.go)

`Interfaces` - End with -er (UserRepository, UserService)

`Constructors` - NewX pattern (NewHandler, NewRouter)