Go with Ent ORM Project Structure

Ent ORM with schema-as-code. Define entities in Go, generate type-safe DB operations.

#go #golang #ent #orm #database #schema #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    main.go  Entry point
    go.mod
    go.sum
    Makefile  generate, migra…
    .env.example
    .gitignore
  cmd/
    api/
      main.go
  ent/  Ent schema and …
    generate.go  go:generate dir…
    client.go  Generated client
    user.go  Generated User …
    user_create.go  Generated build…
    user_query.go  Generated queri…
    schema/  Your schema def…
      user.go  User entity sch…
      post.go
    migrate/  Migration files
      schema.go
  internal/
    handler/
      handler.go
      users.go
    service/
      user_service.go
    router/
      router.go
```

Why This Structure?

Ent is a powerful entity framework for Go. Define schemas in pure Go code, run `go generate`, and get type-safe CRUD operations, eager loading, and migrations. No struct tags, no magic—schemas are explicit code.

Key Directories

`ent/schema/` - Your entity definitions in Go code

`ent/` - Generated client, types, and query builders

`ent/migrate/` - Auto-generated migration files

`internal/service/` - Business logic using ent client

Ent Schema Definition

```go
// ent/schema/user.go
func (User) Fields() []ent.Field {
    return []ent.Field{
        field.String("email").Unique(),
        field.String("name").Optional(),
        field.Time("created_at").Default(time.Now),
    }
}

func (User) Edges() []ent.Edge {
    return []ent.Edge{
        edge.To("posts", Post.Type),
    }
}
```

Getting Started

1. `go install entgo.io/ent/cmd/ent@latest`
2. `ent new User`
3. Define fields and edges in `ent/schema/user.go`
4. `go generate ./ent`
5. Use `ent.Client` in your handlers

When To Use This

- Prefer code-first schema definitions
- Need type-safe query builders
- Complex relationships (edges) between entities
- Want automatic migration generation
- Teams who dislike writing raw SQL

Trade-offs

`Generated code volume` - ent/ directory grows with entity count

`Learning curve` - Ent DSL takes time to master

`Complex queries` - Some advanced SQL requires raw queries

Testing Strategy

`enttest package` - In-memory SQLite client for fast tests

`Integration tests` - Real Postgres with testcontainers

`Schema validation` - ent/schema tests ensure valid definitions