HOW TO ORGANIZE GO CODE

ABOUT ME

Tobiasz Heller

github.com/nephe

Ingrid - backend engineer

AGENDA

- how to organize code into packages
- how to organize application code
- how we do it at Ingrid

PACKAGES

PACKAGES

Package should contain code that has a single purpose.

Follow Unix philosophy to keep things small and use them as building blocks.

```
bytes fmt flag sort sync
```

RELATED PACKAGES

Put related packages into sub-directories

```
net        Package net provides a portable interface for network I/O .
net/http   Package http provides HTTP client and server implementation
net/rpc    Package rpc provides access to the exported methods ...
net/smtp   Package smtp implements the Simple Mail Transfer Protocol
```

PACKAGE STRUCTURE

- Keep tests in the same directory as package code.
- Don't use `src` directory.

```
cost/
  cost.go
  cost_test.go
  testdata/
     testfile1
```

PACKAGE NAMING

- Name should describe purpose of package
- Use simple and clear names (`http, smtp, zip`)
- Use folder hierarchy - `net_http` -> `net/http`
- Don't repeat package name inside types/functions. `cost.CalculateCost()` -> `cost.Calculate()`
- Don't create generic packages (`models, common, utils, helpers`)
- effective Go - package names

APPLICATION

APPLICATION

Code of application/service should be:

- easy to **understand**
- easy to **refactor**
- easy to **test**
- easy to **maintain**

APPLICATION DESIGN

- **monolith** vs **microservice** codebase
- balance between simplicity, speed of development, extensibility, perfection
- too many models and conversions is error prone
- there is no silver bullet
- team should agree what's working best

PACKAGES IN APPLICATION

- **Domain package(s)** define types and interfaces of services and implements basic service.
- **Dependencies package(s)** implementations of domain interfaces grouped per dependency (external data sources, transport logic).

PACKAGES IN APPLICATION

PICKUP POINT SERVICE

```
main.go
pickuppoint/
    location.go
    service.go
    service_test.go
rpc/
    conv.go
    service.go
    service_test.go
store/
    queries.go
    store.go
```

PACKAGES IN APPLICATION

Domain package

```go
// pickuppoint/location.go
type Location struct {
    ID               string
    GroupID          string
  // ...
    Latitude         float64
    Longitude        float64
}
```

PACKAGES IN APPLICATION

Domain package

```go
// pickuppoint/service.go
type store interface {
    InsertLocation(context.Context, *Location) error
    UpdateLocation(context.Context, *Location) error
    GetLocation(context.Context, string, string) (*Location, error)
}

type geocoder interface {
    geocode(context, req) (resp, error)
}

type service struct {
    store    store
    geocoder geocoder
    ...
}
```

PACKAGES IN APPLICATION

Domain package

```go
// pickuppoint/service.go
func (s *Service) UpsertLocation(ctx context, l *Location) error {
    oLoc, err := s.store.GetLocation(ctx, l.GroupID, l.ExternalId)
    if err != nil {
        if errors.Cause(err) != ErrNoLocation {
            return err
        }
        s.fillCoordinates(ctx, l)
        return s.store.InsertLocation(ctx, l)
    }
    if locationAddressEquals(l, oLoc) || oLoc.isForced() {
        copyCoordinates(l, oLoc)
    } else {
        s.fillCoordinates(ctx, l)
    }
    return s.store.UpdateLocation(ctx, l)
```

PACKAGES IN APPLICATION

Transport package gRPC

```go
// rpc/service.go
func (...) UpsertLocation(ctx, req ...) (*empty.Empty, error) {
    if err := req.Validate(); err != nil {
        return nil, grpcerr(codes.InvalidArgument, err)
    }
    loc := locationToModel(req.GetLocation())
    if err := s.backend.UpsertLocation(ctx, loc); err != nil {
        if errors.Cause(err) == pickuppoint.ErrGeoMismatch {
            return nil, grpcerr(codes.InvalidArgument, err)
        }
        return nil, grpcerr(codes.Internal, err)
    }
    return &empty.Empty{}, nil
}
```

PACKAGES IN APPLICATION

For simple CRUD service it is fine to use domain types in storage package.

```
// pickuppoint/location.go
type Location struct {
    ID               string           `db:"id"`
    GroupID          string           `db:"group_id"`
    ...
    Latitude         float64          `db:"latitude"`
    Longitude        float64          `db:"longitude"`
    Metadata         Metadata         `db:"metadata"`
}
type Metadata map[string]string
func (m *Metadata) Scan(src interface{}) error {
    return db.UnmarshalJSON(src, m)
}
func (m Metadata) Value() (driver.Value, error) {
    return json.Marshal(m)
}
```

PACKAGES IN APPLICATION

Storage package

```go
// store/store.go
func (...) InsertLocation(ctx context, l *ppt.Location) error {
    tx, err := s.DB.BeginTxx(ctx, nil)
    if err != nil {
        return errors.Wrapf(err, "failed to begin transaction")
    }
    defer tx.Rollback()
    if _, err = tx.NamedExecContext(ctx, queryInsertLocation, l); err
        return errors.Wrapf(err, "failed to insert location")
    }
    ...
}
```

PACKAGES IN APPLICATION

Main

```go
// main.go
func main() {
    var cfg config
    if err := envconfig.Process("", &cfg); err != nil {
        log.Fatal(err)
    }
    geo := dialGeo(cfg),
    store := store.New(dialDB(cfg))
    backend := pickuppoint.New(store, geo)
    rpcService := rpc.New(backend)

    err := grpc.Serve(rpcService)
}
```

PACKAGES IN APPLICATION

TESTING

- test public API
- define interfaces in package where you are using it
- easy to mock dependencies

OTHER

- use cmd directory if multiple apps from single codebase

```
cmd/
      cmdA/main.go
      cmdB/main.go
```

- `pkg` directory is where you can put your public libraries
- `internal` package can only be imported from its parent directory

SUMMARY

- Focus on simplicity and readability
- Microservice is already scope to one domain
- Balance between perfection and speed
- Group packages by dependency
- Use dependency injection

SOURCES

- effective Go - package names
- Ashley McNamara + Brian Ketelsen - Go best practices
- Ben Johnson - Standard package layout
- Ben Johnson - Structuring applications in Go
- Peter Bourgon - Go best practices 2016
- Mat Ryer - How I write Go http services after 7 years

THANK YOU