Beautify your Golang project

itnext.io/beautify-your-golang-project-f795b4b453aa

In my last article, I have created a service which was unorganized and only for beginner. My motive was to provide a basic concept of creating REST service and go forward step by step.

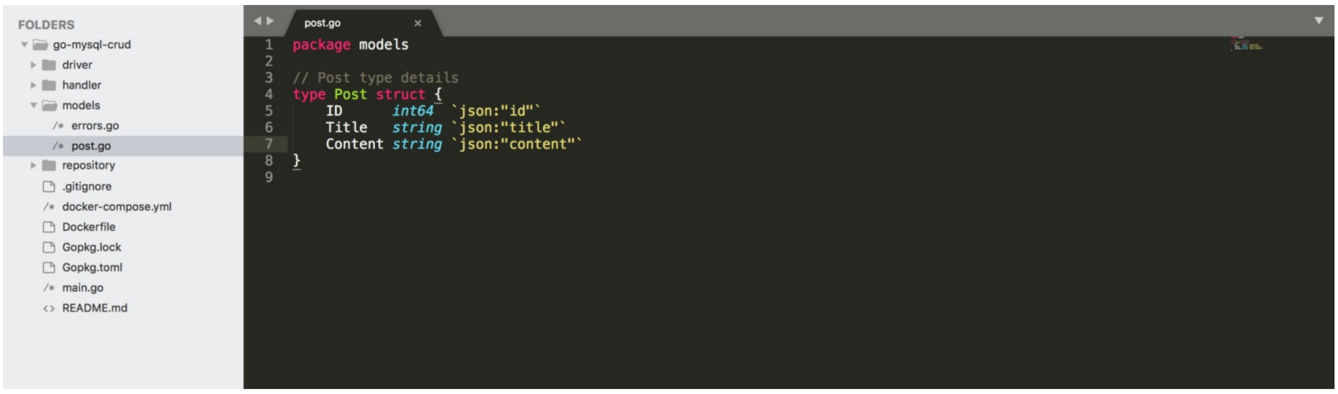
If you are not familiar with my previous post you can go through the link below for understanding functionality. It's not hard to check a glance or if you have previous knowledge about building web API service then you can be processed.

In this tutorial, I am going to show how to organize a project. Organize your code is important because when you want to change some logic of your previous code then the well structured project can easily adopt the changes and as well as save times. Although there are contradictory opinion about project structuring and I prefer the structure of 4 layers. You can replace or modify anything from here.

```
project/
   model/
   repository/
   handler/
   driver/
   main.go
```

models

This layer will store our all model struct and is used by all other layers. Now we will move our `post` model struct on models/post.go file. If we have another model like `author` then those model will also be included in this layer. We have also added error.go file.



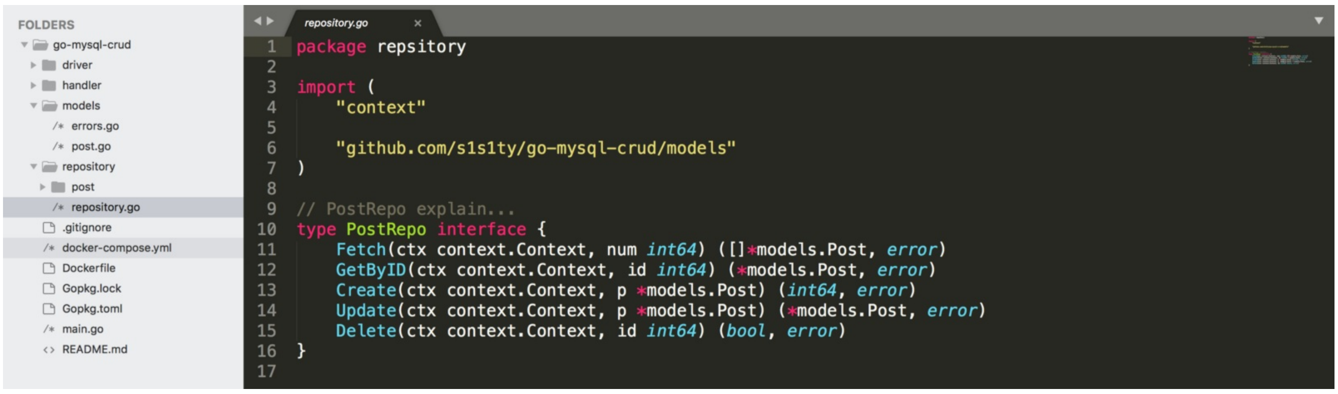
```go
package models

// Post type details
type Post struct {
	ID      int64  `json:"id"`
	Title   string `json:"title"`
	Content string `json:"content"`
}
```

post model

error.go contains your newly created error. You can get the details here.

repository

The repository is responsible for database related job such as querying, inserting/storing or deleting. No business logic is implemented here.



```go
package repsitory

import (
	"context"

	"github.com/s1s1ty/go-mysql-crud/models"
)

// PostRepo explain...
type PostRepo interface {
	Fetch(ctx context.Context, num int64) ([]*models.Post, error)
	GetByID(ctx context.Context, id int64) (*models.Post, error)
	Create(ctx context.Context, p *models.Post) (int64, error)
	Update(ctx context.Context, p *models.Post) (*models.Post, error)
	Delete(ctx context.Context, id int64) (bool, error)
}
```

post repo

Here, I create repository.go inside the repository folder. This file is indebted to hold all your repository related interface. If we have an additional domain for instance author or something then all author method within an interface is included here.

If you notice the left side of the image above, I have created a folder, named post. We'll implement our interface on `post/post_mysql.go` file. Why is suffix MySQL? Because if we've another database for post repository like mongo or redis then we'll include post_mongo.go(whatever you call) and implement mongo related query.

post_mysql.go must satisfy the PostRepo interface. If we want to add any extra method then we have to include it to the interface and implement it on post_mysql.go file.

handler

Basically handler folder can confide to the repository as this layer is decided which repository layer will use. Any process is handled here. This layer accepts the request, call the repository layer and satisfy the business process and send the response.

Handler layer also contains several protocols implementation like RPC, REST etc and isolated with a different folder.

driver

In our structure, the driver layer is described by all our database connections. This layer is responsible for connecting with the database and send connection object to the controller.

Here, We'll use main.go instead of the controller as our service has only one domain with CRUD operations.

From our main.go, we'll provide all database credential and connect with database. We read our DB credential in environment variable which is provided on docker-compose file. We also keep our all route here but you can isolate it.

The Sample Projects

You can check the sample project here

https://github.com/s1s1ty/go-mysql-crud

I also encourage you to add test file. Again, Any design is not like the Bible, We have to implement it according to our project definition, business logic etc.

In Conclusion

You can learn more about this topic via the links below.

> Anything related to this post you can ask me on comment section or

At last, You can shoot me the comment section below if you notice any mistake I did. You can also send me PR on my GitHub **repository.**