1. Introduce your self:

Hello, my name is Thang

I have a **software enginee degree** and have about **1 years experience** as a **golang developer.**

My background is a **back-end developer**

I used to work with C# in ASP.NET but I have switched to **Golang** and spent most of my time in Golang projects.

I also have experience in front-end using VueJS. And familiar with noSQL and SQL databases for example mongoDB, postgresSQL.

I'm currently working in a startup company call Jamalex.

Here I have participated in many projects, most of them are e-commerces applications. Because it a small company so I have played many roles from structure projects, designing databases to creating services.

Before here, I have worked in TMA Solution for about 1 year but during the covid 19, there wasn't a lot Golang jobs for me so I decided to move on.

During the time working for TMA, have participate into 2 projects:

- First one is an **education project** for **university**, the team size is around 10 people and my role was using **ASP.NET MVC 5** framework to create services.
- Second one is a **golang project** that create an aplication for project manager. Because it's a small team so I have play many roles from **structure the project** from begining to **analys database** and **create services**.
- Technology use in project are **Golang**, **VueJS**, **MongoDB**, **Docker** and **ElasticSearch**

Also I'm confidence in my communication skill and really want to work with internation team.

So I hope that I can join your team !

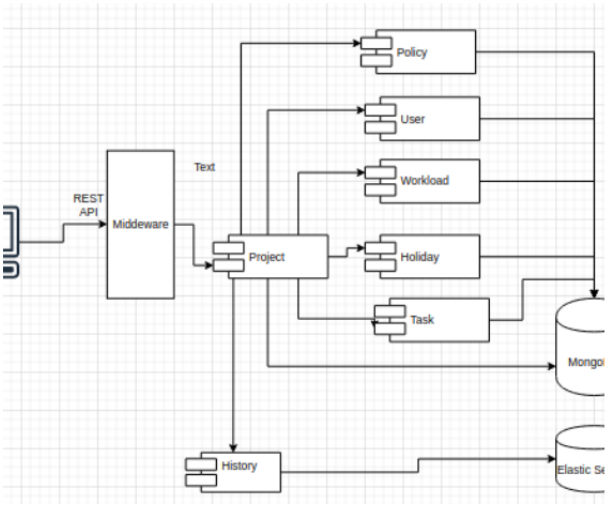
2. Golang project in details:

The main requirement is to create an application for manager to keep track on their project status.

By breaking down a big project in to smaller task and managing each task.

The application uses a **diagram** and a **calendar** to show the task infomation to users.

Used VueJS in front-end, Rest API to communicate with Golang web server, MongoDB to store data, Docker for deployment and Elasticsearch as a search engine.

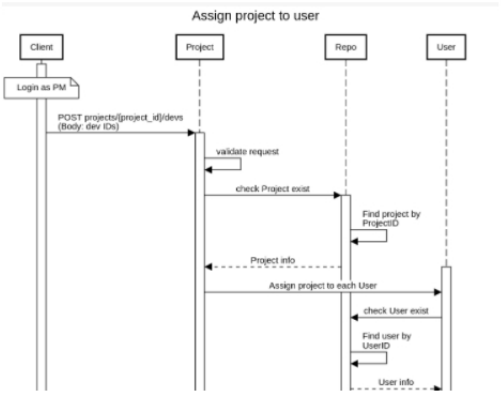


With those requirements, the application is divided into **8 moludes** and have worked with **4 main modules:**

- The most important is the **Project module**, where **I manage** all the **actions** inside a **project** include create tasks, view history, manage developers, or views holiday
- **Second** is the **User module** where I manage all the users inside the application
- **Third** is the **History module** where all the actions in a project will be saved and strore to **ElasticSearch** so that users can easily search and go back an older version of the project.
- Last but not least is the **Workload module**, in here I calculate all the amount of work for each developers so that users can dicide should or shouldn't assign task to that developer.
- I also took part in the **Authentication** & **Authorization** module. For **authentication** I use the **JWT mechanism** and for **authorization** I use the role base access control (**RBAC**) model

3. How a module work:

- In each module, there are three layer **handler**, **service** and **repository**
- When a request came from client, server with route the request to a specific **handler**
- **If there is any authentication require,** there will be a middle-ware to check if that request is **authenticated** or not.
- After that **handler layer** will call the needed **service**
- The **service layer** will do the business logic: valid or transform data
- Then the service will call **repository layer** to interact with database
- And finally, the **handler** with response to client with data and messages



Real experience questions :

1. How do you structure your Go project?

I follow the **Standard Go Project Layout,**

- **First,** I used **Go Module** to manage all the dependencies
- Then I created 4 folders:

- **Config folder**: will contains some configs files for the application.

- **Deployment folder**: will contains all configuration files for deployment. It has file to deploy the application at different enviroment.

- **Web folder**: will have all files for front-end

- **Internal folder**: will contain all the code for backend. The main purpose of this folders is that it's a private folder, no one can import this code into their applications. This is the most important folder. It have 2 sub folders that are:

  - Pkg folder : contains the external library that I want to customize before use inside my application. For example **mgo**, **gorilla-mux**, **jwt or logrus**.

  - App folder: contains all code for server. Each module of the application is inside a seperate folder. For example:

  + Code for the **authentication** will be inside the **auth** folder, code for **user** module will be inside the **user** folder. The application have eight module, so there will be 8 eight folders for each module

  + Inside each folder, there will be **3 main files**:

  - A **handler file to** handle requests and responses

  - A **service file**: contains code for business logic

  - A **repository file** to interact with database

  - I also have **test file** inside these folder

- Beside those 4 folder, I keep all others none go file in the root directory: for example makefile, README, .gitignore, go.mod, go.sum,...

Advantages:

It helps manage applications more efficiently at **scale**. It structured in a way that It can change into Microservice **Architecture** by separate each module into an independent services without affect others module.

2. Authentication: JWT

For authentication, I use the **IWT** which is one of the **token based authentication** mechanism.

To use the **JWT** in golang project, I use the **jwt-go** library. It have good document, easy to customize and use in Go project.

- So when a user submits username and password for login
- Server will validates and returns a signed JWT token.
- User then will use that token for future requests
- When the request came to the server, If there is any authentication require, there will be a middle-ware to check if that request is **authenticated** or not.

The advantages of JWT is that server now can include some information in the token, and JWT works well on micro-services architecture.

3. Authorization: Role based access control

- For authorization I used the **Role based access control (aka RBAC)** model which restrict permission of user base on there role inside the project.
- To implement **RBAC**, I use the **casbin lib** which is a powerful access control library for Golang projects.
- With casbin, it very easy to define policies for users. That means a user with a **specific roles** can access a specific **data**
- I only need one **configuration file** and a **document** in database to store all of the policies for users.
- And the library also support a **adapter** to connect to **mongoDB**, so I can easily create, update or delete policy for users.

4. Database:

Have you ever used MongoDB? Explain in detail what did you use it for?

- MongoDB is a noSQL database. It helps build applications faster, can handle many data types and manage applications more efficiently at scale.
- I used the latest version (4.2) of mongoDB to store all data of the application.

Explain what are the differences between NoSQL and SQL? When should I use NoSQL and when should not?

- Non-relational and relational databases, document, collections instead of tables and relation ships between tables
- SQL is designed to scale up, NoSQL is designed to scale out
- Use noSQL when: constantly adding new features, have a lot of data, many different data types, need to scale in future
- Not use noSQL when: need complex queries, not working large volume of data or many data types.

What library do you use to interact with MongoDB in Go?

- Use the mgo lib as a mongoDB driver for Go. With mgo I can easily download, modify and use it for the application.With a few step I can create a Session and connect to mongodb.

Compare MongoDB official lib with mgo?

- mgo (community version) with simple syntax, easy to use with big community support.
- mgo use one master session, copy it when want to use and close after done.

How do you scale your MongoDB for performance?

- Vertical scaling: add more resource to mongo server.
- Horizontal scaling: MongoDB provides sharding mechanism. Distribute the write/ load across multiple servers(shards)

5. ElasticSearch:

What did you use Elasticsearch for?

ES is a full-text search engine, it's a version of the Lucen search framework which used indeces to search. ES can store and search a huge amounts of data quickly and in near real time.

I use elastic search to store all the history of a project. Because the application have a large volume of history's record, and need to search every detail of the history.

When should you use Elasticsearch and when should you use MongoDB?

Shouldn't use ES for main database (can lost write data), mongoDB is better for few indeces. Use es for search engine if need to digging though the specific detail of the data for analys

What library do you use to interact with Elasticsearch in Go?

It also easy to use and implement into Go project. I used a olivere/elastic library recommend on go-awesome to implement ES to the applications.

How do you handle a lot of indexing in high performance application?

Have good index congiguration, have a suitable amout of shards and replica

I also use Kibana on top of ElasticSearch for the purpose of testing and visualize indeces.

6. Deployment:

How do you deploy your application?

Use docker to deploy the application

There are two different type of deployment inside my application. One for development and another one for production.

In develop environment, I used a library call realize to support live-reload server.

In the production, after built the application into a binary file. I write a dockerfile to build a new image that include an alpine image and my binary file running inside

Then I can run that image to get a container with docker-compose. Because the application uses multiple services (mongoDB, elasticsearch, kibana,...)

7. Config

To manage configurations for the application, I use the envconfig library recomment on go-awesome.

I can load configurations from enviroment and the library also support for default enviroment variables with structure tag.

With Docker, for each enviroment have a seperate .env file to content all the enviroment variable for each service. This file is use to map dollar-notation variables in docker-compose file in the same folder.

8. HTTP

Built in net/http package:

- HandlerFunc
- ListenAndServe
- Handler: func(ResponeWrite, *Request)
- DefaultServerMux

Third-3rd package: gorilla/mux

- famous community lib, easy to use, with simple syntax, easy to find help,..
- compatile with the standard library
- Request can have prefix, schema, query values with regular expression
- support middlerware & graceful shutdown
- Middlewares are small pieces of code which take one request, do something with it, and pass it down to another middleware or the final handler.

9. Logging:

Only use the logrus library to log out the information of the appication at different format. I include it into the application and customize it for a better format with different level log and color.

10. How did you test your application?

For intergration test, used **Postman** to test all the APIs that the application has.

And for unit-test, I used the built-in **testing** package to test the service layer of each module. Also I used **gomock** to generate fake data for the interfaces that the service has.

Also I used the go test coverage option and output the result to a html file to view how much of the code I have cover.

11. Performance

How can you measure performance of your application?

- To measure performance I used the built-in **benchmark** function in the testing package. With **benchmark** I can see how long it took (in **nanosecond**) to run an operations. It can write a benchmark in the same file with unit-test and follow the rule of unit-test.
- But benchmark didn't have enough information to measure perfomance, so used a HTTP benchmarking tool. With this tool, I can set the **number of threads**, number of **HTTP connections** and how long I want to run the benchmark.
- After that, in the result, I can see how many request per second that API can have and see the latency. For example: in my applicaion, I ran the tool with a **GET** request on **6 threads** & **400 connections**. The outout give **24k**req/s with latency nearly **15ms**

How do you scale up your application?

- Add more resource for the server, more cpu, memory, storage, or network can be added to that system to keep the performance at desired levels
- Scale out database, scale out server, use load balancing.

Talk about the challenges that you did:

Questions about Go:

- Compare Go vs Java, C, Python, C++, Java scripts:

Go is an oper source programming languages that makes it easy to build simple, reliab le, and efficient software.

- Goroutine vs Thread:

- Array vs slice:
  - Collection of items have same type
  - Length, comparablerity, reference

array: lengh is fix, cannot change, reference: pass by value, not effect the original, can compare 1 array with another

slice: can resize, built on top of a array ,pass reference of the underlying array. Can't compare.

Slice inclue: data (pointer to an array), lenght, capacity

- String:
  - Read only slice of byte (rune), UTF-8
  - immutable string
- What is a worker pool? And how to do that in Go?
- What is fan-in, fan-out?