API write in Go

Pallat Anchaleechamaikorn

Technical Coach at Infinitas by KrungThai

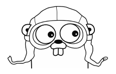
yod.pallat@gmail.com

https://github.com/pallat

https://dev.to/pallat

https://go.dev/tour (Thai)

https://github.com/uber-go/guide (Thai)



Outline

- Gorm (ORM) library
- REST API with net/http
- Gin-Gonic web-framework
- JWT
- Go Middleware
- Service Configurations
- Gracefully Shutting Down
- Dockerfile

database/sql

```go
import (
    "database/sql"
    _ "github.com/mattn/go-sqlite3"
)
```

the Blank identifier

(underscore)

import with need the side effects

_ "github.com/mattn/go-sqlite3"

`_` Blank identifier

```
import (
    _ "myproject/effect"
)
```

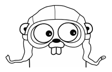
we don't want to use any stuff from that package

we need just the effect from `init()` in there

database/sql with sqlite

```go
func main() {
    db, err := sql.Open("sqlite3", "todo.db")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    // TODO: logic
}
```



Connect to Database

> /play/database

https://github.com/mattn/go-sqlite3/blob/master/_example/simple/simple.go

Change foo to todo

> foo.db -> todos.sqlite

```
create table todos (id integer not null primary key, task text, done integer, dttm integer);
delete from todos;
```

```
insert into todos(id, task, done, dttm) values(?, ?, ?, ?)
```

net/http

```go
package main

import (
    "io"
    "log"
    "net/http"
)

func main() {
 // Hello world, the web server

 helloHandler := func(w http.ResponseWriter, req *http.Request) {
    io.WriteString(w, "Hello, world!\n")
 }

    http.HandleFunc("/hello", helloHandler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Requirement

Todo List API

> **POST** */todos*
>
> **GET** */todos*
>
> **PATCH** */todos*
>
> **DELETE** */todos/:**id***

Start with net/http

init the project

```
mkdir todoapi && cd todoapi
git init
go mod init github.com/pallat/todoapi
```

gorilla/mux

https://pkg.go.dev/net/http

https://github.com/gorilla/mux

Web Framework Benchmark

https://www.techempower.com/benchmarks/#section=data-r19&hw=ph&test=plaintext

https://github.com/smallnest/go-web-framework-benchmark

Fiber

https://docs.gofiber.io/

https://github.com/gofiber/fiber

Fiber: Example

```go
package main

import "github.com/gofiber/fiber/v2"

func main() {
    app := fiber.New()

    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("Hello, World 👋!")
    })

    app.Listen(":3000")
}
```

Fiber.io

★ Some values returned from *fiber.Ctx are not immutable by default

Issue

https://github.com/gofiber/fiber/issues/426

Object-Relational Mapping (ORM)

gorm.io

xorm.io

ent (entgo.io)

gorp

etc.

https://github.com/avelino/awesome-go#orm

Gorm.io

```go
import (
    "gorm.io/gorm"
    "gorm.io/driver/postgres"
)

func main() {
    dsn := "host=localhost user=postgres password=mysecretpassword dbname=myapp port=5432"
    db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
    if err != nil {
    panic("failed to connect database")
    }

    // TODO: logic
}
```

Gorm Model

```go
type Model struct {
    ID        uint `gorm:"primarykey"`
    CreatedAt time.Time
    UpdatedAt time.Time
    DeletedAt DeletedAt `gorm:"index"`
}
```

Composition Gorm Model, embeded field

```go
type Todo struct {
   gorm.Model
   Task string  `json:"task"`
   Done bool    `json:"done"`
}
```

Todo Package

- Json binding
- Insert into Table
- Return Json error
- Return OK Response

Gin-Gonic

https://github.com/gin-gonic/gin

recommended by golang.org

Tutorials

https://golang.org/doc/tutorial/

Developing a RESTful API with Go and Gin

https://golang.org/doc/tutorial/web-service-gin

Gin-Gonic Example

```go
package main

import "github.com/gin-gonic/gin"

func main() {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
    })
    r.Run() // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")
}
```

Gin Handler

```go
type HandlerFunc func(*Context)
```

usage

```go
func pingPongHandler(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
}
```

Gin Context

```go
type Context struct {
    Request *http.Request
    Writer  ResponseWriter

    Params Params

    // Keys is a key/value pair exclusively for the context of each request.
    Keys map[string]interface{}

    // Errors is a list of errors attached to all the handlers/middlewares who used this context.
    Errors errorMsgs

    // Accepted defines a list of manually accepted formats for content negotiation.
    Accepted []string
    // contains filtered or unexported fields
}
```

Let's start with New Todo Handler

```
POST /todos
Content-Type: application/json

{
    "task": "Daily Sync-up"
}
```

JWT

jwt.io

Standard Claims

"iss" (Issuer)

"sub" (Subject)

"aud" (Audience)

"exp" (Expiration Time)

"nbf" (Not Before)

"iat" (Issued At)

"jti" (JWT ID)

Protect API

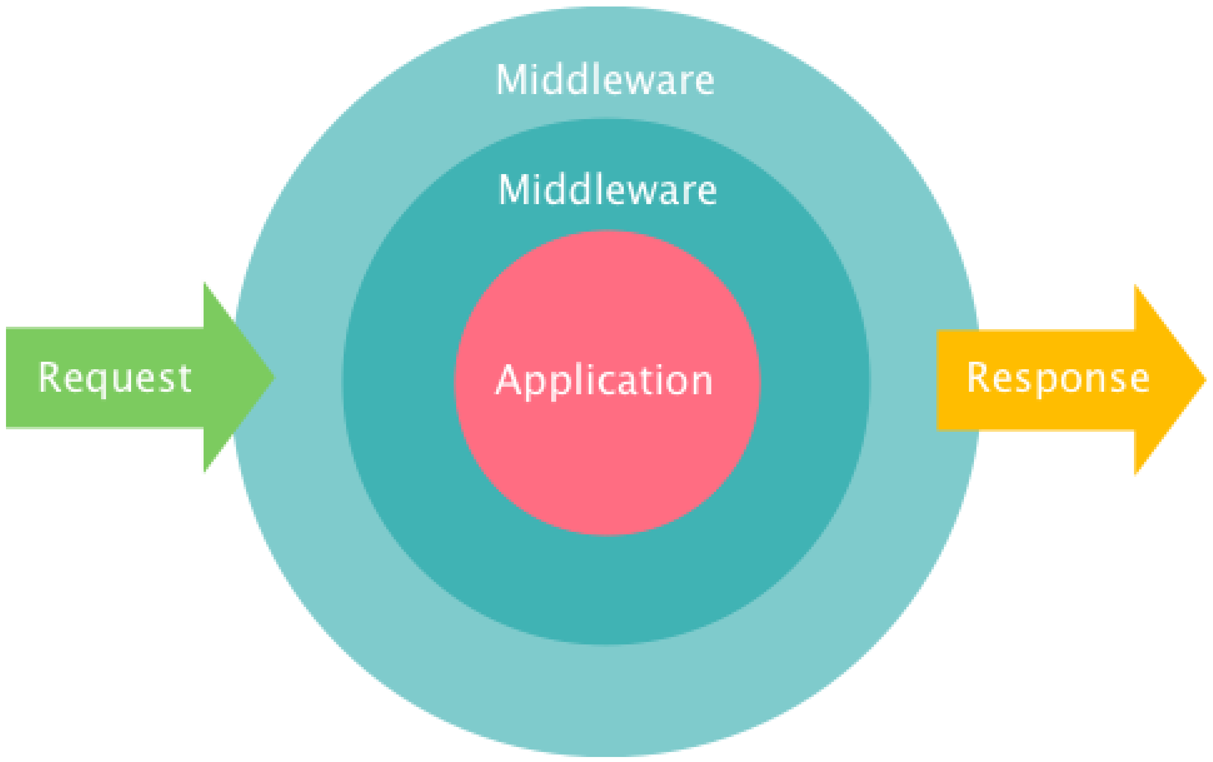
POST /todos

Content-Type: application/json

Authorization: *Bearer*
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6I
kpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk
6yJV_adQssw5c

```
{
    "text": "Daily Sync-up"
}
```

Middleware



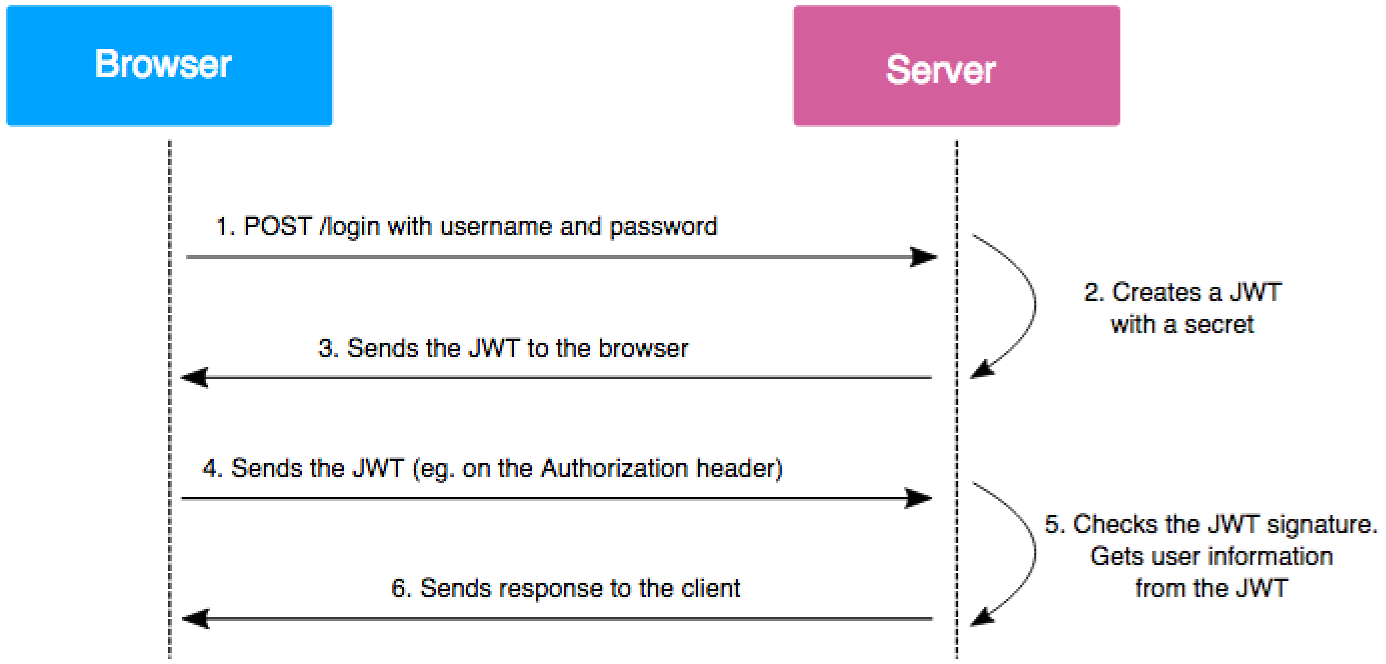
Gin Middleware chain

```
c.Next()
```

beak the chain by Abort

```
c.AbortWithStatus(http.StatusUnauthorized)
```

JWT Middleare


Browser
Server
1. POST /login with username and password
2. Creates a JWT with a secret
3. Sends the JWT to the browser
4. Sends the JWT (eg. on the Authorization header)
5. Checks the JWT signature. Gets user information from the JWT
6. Sends response to the client

Login

POST /login

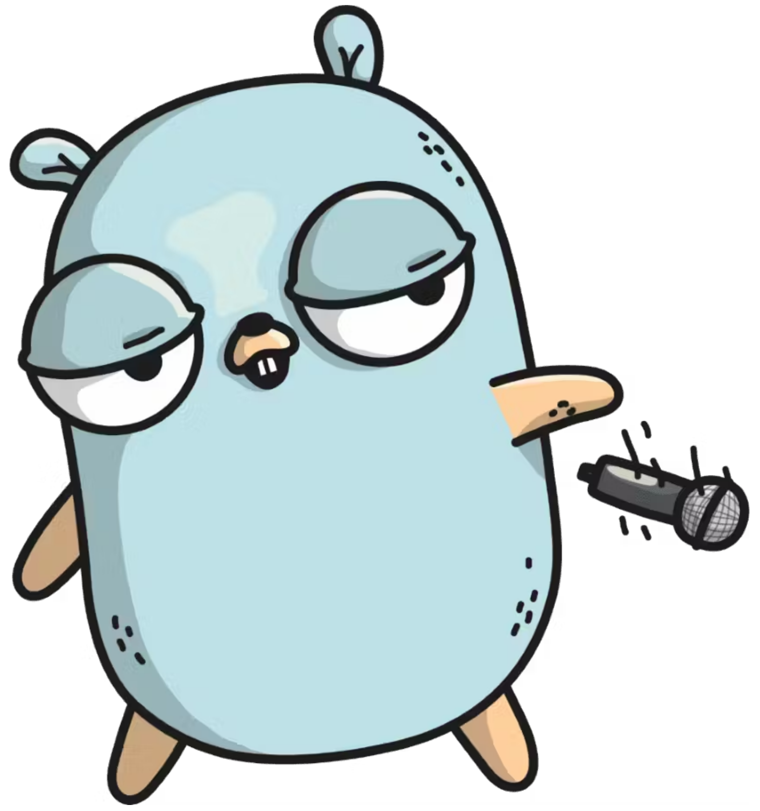
Content-Type: application/json

```
{
    "account": "pallat",
    "password": "drowssap"
}
```

Swagger

https://github.com/swaggo/gin-swagger

Gracefully shutting down



ListenAndServe

image

The Kubernetes termination lifecycle

https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-terminating-with-grace

`In practice, this means your application needs to handle the SIGTERM message and begin shutting down when it receives it. This means saving all data that needs to be saved, closing down network connections, finishing any work that is left, and other similar tasks.`

graceful example

```
ctx, stop := signal.NotifyContext(context.Background(), os.Interrupt)
defer stop()

go func() {
    if err := server.ListenAndServe(); err != nil && err != http.ErrServerClosed {
        log.Fatalf(“listen: %s\n”, err)
    }
}()

<-ctx.Done()
stop()
fmt.Println("shutting down gracefully, press Ctrl+C again to force")

timeoutCtx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()

if err := server.Shutdown(timeoutCtx); err != nil {
    fmt.Println(err)
}
```

test shutdown

```
sudo lsof -i :8080
kill -15 [PID] || kill -SIGINT [PID]
kill -SIGTERM [PID]
```

Configurations

https://12factor.net/

Viper

https://github.com/spf13/viper

.env auto loading

https://github.com/joho/godotenv

Our Config

.env

```
ADDR=":8081"
SIGNATURE="drowssap"
```

Add external to the binary

```
go build -ldflags “-X main.somevar=training” -o app
```

Add git commit number to the binary

```
go build \
  -ldflags "-X main.buildcommit=`git rev-parse --short HEAD` \
  -X main.buildtime=`date "+%Y-%m-%dT%H:%M:%S%Z:00"`" \
  -o app
```

Dockerfile

```
FROM golang:1.20-buster AS build
WORKDIR /app
COPY go.mod ./
COPY go.sum ./
RUN go mod download
COPY . ./
ENV GOARCH=amd64
RUN go build -o /go/bin/app

## Deploy
FROM gcr.io/distroless/base-debian11
COPY --from=build /go/bin/app /app
EXPOSE 8081
USER nonroot:nonroot
CMD ["/app"]
```

Docker build

```
docker build -t todo:test -f Dockerfile .
```

Docker run

```
docker run -p:8081:8081 --env-file ./.env --link some-mariadb:db --name myapp todo:test

docker run --name some-postgres -p 5432:5432 -e POSTGRES_PASSWORD=mysecretpassword -e POSTGRES_DB=myapp -d postgres
```