Golang Programming Workshop
Web API apps

Contents

1 API service

2 Database Access

What a REST service is without a database, let's do it.

2.1 Postgres

Package `database/sql` provides generic interface for SQL databases. In our exe

1. Prepare the project

```
# anywhere
$ mkdir workshop-db
$ go mod init github.com/wojciech12/workshop-db
$ go get github.com/lib/pq
$ go get github.com/jmoiron/sqlx
```

2. Run psql:

```
# user: postgres
$ docker run --rm \
  --name workshop-psql \
  -e POSTGRES_DB=hello_world \
  -e POSTGRES_PASSWORD=nomoresecret \
  -d \
  -p 5432:5432 \
  postgres
```

Notice:

```
$ psql hello_world postgres -h 127.0.0.1 -p 5432
```

3. Connect to db:

```
package main

import (
  "database/sql"
  "fmt"
  "net/url"

  _ "github.com/lib/pq"
)

var driverName = "postgres"

func New(connectionInfo string) (*sql.DB, error) {
  db, err := sql.Open(driverName, connectionInfo)
  if err != nil {
    msg := fmt.Sprintf("cannot open db (%s) connection: %v",
      driverName, err)
    println(msg)
    return nil, err
  }
  return db, nil
}

func main() {
  user := url.PathEscape("postgres")
  password := url.PathEscape("nomoresecret")
  host := "127.0.0.1"
  port := "5432"
  dbName := "hello_world"
  sslMode := "disable"

  connInfo := fmt.Sprintf(
    "postgres://%s:%s@%s:%s/%s?sslmode=%s",
    user, password, host, port, dbName, sslMode)

  sql, err := New(connInfo)
  if err != nil {
    panic(err)
  }
  err = sql.Ping()
  if err != nil {
    panic(err)
  }
  defer sql.Close()
```

```
}
```

4. Let's create tables using the following definition:

```
CREATE TABLE users (
  id BIGSERIAL PRIMARY KEY,
  first_name TEXT,
  last_name TEXT);
```

5. Create table in Golang:

```
func createTableIfNotExist(sql *sql.DB) {
  _, err := sql.Exec(`CREATE TABLE users (
    id  BIGSERIAL PRIMARY KEY,
    first_name TEXT,
    last_name TEXT)`)
  fmt.Printf("%v\n", err)
}
```

6. Add lines:

```
func insertData(sql *sql.DB, firstName string,
    lastName string) error {
  iq := `INSERT INTO users (first_name, last_name)
       VALUES ($1,$2) RETURNING id;`
  stmt, err := sql.Prepare(iq)
  if err != nil {
    return err
  }
  defer stmt.Close()
  _, err = stmt.Exec(firstName, lastName)
  if err != nil {
    return err
  }
  return nil
}
```

6. Read lines:

```
func readData(sql *sql.DB) error {
  s := `SELECT id, first_name, last_name FROM users`
  rows, err := sql.Query(s)
  if err != nil {
    return err
  }
  defer rows.Close()

  type person struct {
    ID         int
    FirstName  string
    SecondName string
  }

  var p person
  for rows.Next() {
    if err := rows.Scan(
      &p.ID,
      &p.FirstName,
      &p.SecondName); err != nil {
      return err
    }
    fmt.Printf("%d %s %s", p.ID, p.FirstName, p.SecondName)
  }
  return nil
```

7. With sqlx[1], you can have more declarative code for working with your database:

```
dbx := sqlx.NewDb(sql, driverName)
```

```
func insertData2(sql *sqlx.DB, firstName string,
  lastName string) error {
  type input struct {
    FirstName string `db:"first_name"`
```

[1] https://github.com/jmoiron/sqlx

```
    LastName  string `db:"last_name"`
  }
  type output struct {
    ID int64 `db:"id"`
  }

  var out output
  var in input

  in.FirstName = firstName
  in.LastName = lastName

  sqlQuery := `INSERT INTO users ( first_name,
            last_name
           ) VALUES (
         :first_name,
         :last_name) RETURNING id`

  stmt, err := sql.PrepareNamed(sqlQuery)
  if err != nil {
    return err
  }
  err = stmt.Get(&out, in)
  if err != nil {
    return err
  }
  fmt.Println(out.ID)
  return nil
}
```

Notice: for select queries, you use `Queryx` and `err := rows.StructScan(&out)`.

8. Add support for the database in your web app.

2.2 Database migrations

We will not cover the database migrations in this workshop.

Check golang-migrate/migrate[2].

[2] https://github.com/golang-migrate/migrate

2.3 Testing your database integration

In the Golang community, we test against real databases if we can. The best practice is to use build tags to distinguish integration tests:

```
// +build integration

package service_test

func TestSomething(t *testing.T) {
  if service.IsMeaningful() != 42 {
    t.Errorf("oh no!")
  }
}
```

To run:

```
$ go test --tags integration ./...
```

2.4 GORM

github.com/go-gorm/gorm

2.5 Mongodb

A homework, prepare an application that uses mongodb as its database: Database:

```
$ docker run  -p 27017:27017 \
  --name da-mongo \
   -d \
   mongo
```

Let's setup our project:

```
# anywhere
$ mkdir workshop-mgo
$ go get github.com/globalsign/mgo
```

3 Best practises

1. Dependencies Injection, without the magic:

```go
func main() {
    cfg := GetConfig()
    db, err := ConnectDatabase(cfg.URN)
    if err != nil {
        panic(err)
    }
    repo := NewProductRepository(db)
    service := NewProductService(cfg.AccessToken, repo)
    server := NewServer(cfg.ListenAddr, service)
    server.Run()
}
```

2. Dependencies direction from supporting pkgs to business logic pkgs.

3. `Context`, pass to all the functions.

4 Observability

4.1 Monitoring with Prometheus

See https://github.com/wojciech12/talk_monitoring_with_prometheus

4.2 Logging with Logrus

Example for a talk on logging[3]

```go
package main

import (
  "fmt"
  "net/http"

  "github.com/gorilla/mux"
```

[3] https://github.com/wojciech12/talk_observability_logging

```
   log "github.com/sirupsen/logrus"
)

func HelloHandler(w http.ResponseWriter, r *http.Request) {
  w.WriteHeader(http.StatusOK)
  fmt.Fprintf(w, "Hello!")

  log.WithFields(log.Fields{
      "method": r.Method,
      "handler": "hello",
  }).Info("hello!")
}

func WorldHandler(w http.ResponseWriter, r *http.Request) {
  w.WriteHeader(http.StatusOK)
  fmt.Fprintf(w, "World!")

  log.WithFields(log.Fields{
      "method": r.Method,
      "handler": "world",
   }).Info("world!")
}

func ErrorHandler(w http.ResponseWriter, r *http.Request) {
  w.WriteHeader(http.StatusOK)
  fmt.Fprintf(w, "Bye!")

  log.WithFields(log.Fields{
    "method": r.Method,
    "handler": "error",
  }).Error("What does 'bye' mean?!")
}

func main() {

  log.SetFormatter(&log.JSONFormatter{})

  r := mux.NewRouter()
  r.HandleFunc("/hello", HelloHandler)
  r.HandleFunc("/world", WorldHandler)
  r.HandleFunc("/error", ErrorHandler)
  http.ListenAndServe(":8080", r)
}
```

See also https://martinfowler.com/articles/domain-oriented-observability.html for a discussion on how and what to monitor.

4.3 Tracking

TBA

4.4 Open telemetry

TBA

5 Tools

5.1 goreleaser

A very sharp tool that greatly simplifies your CI/CD pipeline for Golang apps.

```
project_name: myapp
release:
  github:
    owner: YOUR_USER_OR_ORG
    name: myapp
  name_template: '{{.Tag}}'
builds:
- env:
  - CGO_ENABLED=0
  goos:
  - linux
  goarch:
  - amd64
  main: .
  ldflags: -s -w -X main.version={{.Version}} -X main.commit={{.Commit}} \
    -X main.date={{.Date}}
  binary: myapp
archive:
  format: tar.gz
  name_template: '{{ .ProjectName }}_{{ .Version }}_{{ .Os }}_{{ .Arch }}{{ if .Arm
    }}v{{ .Arm }}{{ end }}'
snapshot:
  name_template: snapshot-{{.ShortCommit}}
checksum:
  name_template: '{{ .ProjectName }}_{{ .Version }}_checksums.txt'
dist: dist
dockers:
  - image: YOUR_USER_OR_ORG/myapp
```

5.2 Docker

- Compile on your machine:
  `GOOS=linux GOARCH=amd64 CGO_ENABLED=0 go build ./...`
  and put just binary inside the Docker

- An alternative is to use multi-stage Docker builds
- Final image `alpine` or `ubuntu`

5.3 Performance tests

My favorite tool:

- https://locust.io/