Criando uma API Rest com Database em Golang

Cleuton Sampaio

Vamos criar uma api bem simples: Um CRUD para guardar dados de candidatos a emprego. Ela permitirá:

- Cadastrar um candidato
- Atualizar dados de um candidato
- Deletar um candidato
- Listar candidatos

A ideia é manter bem simples.

Vamos utilizar o pacote "net/http" e o projeto "golang-migrate" para criar e manter um database PostgreSQL tudo rodando no Docker.

Começaremos com o banco de dados e depois vamos para o código.

Banco de dados

Suba um servidor PostgreSQL com Docker:

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

Substitua "mysecretpassword" por alguma senha sua. Assim, criamos um servidor Postgres e vamos utilizar o database postgres com o usuário padrão e a senha informada.

Crie uma **pasta para o projeto** e crie um módulo **Golang** nela:

```
mkdir [nome]
go mod init com.example/[nome]
```

Escolha um nome para seu projeto (sem expaços) e substitua "[nome]" por ele.

Agora, crie uma pasta "db/migrations". Nesta pasta criaremos as "migrations" para nosso banco de dados. Os arquivos "up" criam tabelas e objetos e os arquivos "down" apagam. Vamos rodar o **golang-migrate** e primeiramente, vamos instalá-lo (você também pode usar Docker, mas fica mais fácil instalando):

MS Windows:

```
scoop install migrate
```

Linux:

```
$ curl -L https://packagecloud.io/golang-migrate/migrate/gpgkey | apt-key
add -
$ echo "deb https://packagecloud.io/golang-migrate/migrate/ubuntu/
$(lsb_release -sc) main" > /etc/apt/sources.list.d/migrate.list
$ apt-get update
$ apt-get install -y migrate
```

MacOS:

```
brew install golang-migrate
```

E vamos criar uma migração para criar as tabelas:

```
migrate create -ext sql -dir db/migrations -seq candidates
```

Dois arquivos serão criados na pasta "db/migrations":

- db/migrations/000001_candidates.down.sql
- db/migrations/000001_candidates.up.sql

Vamos editar o arquivo "up" para criarmos a tabela:

```
CREATE TABLE IF NOT EXISTS candidates (
  id BIGSERIAL primary key,
  name TEXT not null,
  created_at TIMESTAMP default now()
);
```

E, no arquivo "down", vamos criar o código para apagar a tabela:

```
DROP TABLE IF EXISTS candidates;
```

E vamos criar mais migrations, por exemplo, para preencher a tabela com alguns usuários:

```
migrate create -ext sql -dir db/migrations -seq insert-candidates

...
--- _insert-candidates.up.sql ---
INSERT into candidates(name, created_at) VALUES
('Candidate #1', '2022-11-01 10:10:10'),
('Candidate #2', '2022-11-01 10:10:10'),
('Candidate #3', '2022-11-01 10:10:10');
```

```
...
--- _insert-candidates.down.sql ---
DELETE FROM candidates;
```

Agora, vamos popular o banco:

```
migrate -path ./db/migrations -database "postgresql://postgres:
[senha]@localhost:5432/postgres?sslmode=disable" -verbose up
```

Criação do servidor básico

Para este servidor, vamos utilizar o "net/http" com a biblioteca "gorilla/mux". Então, precisamos instalar o "gorilla/mux":

```
go get -u github.com/gorilla/mux
```

E também precisamos instalar o driver SQL do Postgres:

```
go get -u github.com/lib/pq
```

Este driver é utilizado indiretamente pelo pacote "database/sql" queremos importá-lo apenas para inicialização, pois não vamos utilizá-lo diretamente, então o import tem que ser assim:

```
import _ "github.com/lib/pq"
```

Vamos usar "net/http" para criar nosso servidor. A primeira coisa é criar uma pasta "cmd" e, dentro dela, um arquivo "main.go". Você pode ver esse arquivo no repositório.

Estrutura

Começamos criando alguns tipos e funções auxiliares, como a conexão com o database. Nesta função, eu peguei muita coisa das variáveis de ambiente, para evitar hardcoding (eu criei uma função auxiliar para fornecer default):

```
func connectDB() (*sql.DB, error) {
    host := getEnv("DEMO_HOST", "localhost")
    dbPort := getEnv("DEMO_DBPORT", "5432")
    dbUser := getEnv("DEMO_USER", "postgres")
    password := getEnv("DEMO_DATABASE_PASSWORD", "mysecretpassword")
    dbName := getEnv("DEMO_DBNAME", "postgres")
```

```
    sslMode := getEnv("DEMO_SSLMODE", "disable")
    connectString := fmt.Sprintf("host=%s port=%s user=%s password=%s
dbname=%s sslmode=%s",
        host, dbPort, dbUser, password, dbName, sslMode)
    db, err := sql.Open("postgres", connectString)
...
```

E criei uma função para facilitar a vida ao escrever as respostas HTTP dos nossos handlers:

```
func WriteResponse(status int, body interface{}, w http.ResponseWriter) {
    w.WriteHeader(status)
    w.Header().Set("Content-Type", "application/json")
    if body != nil {
        payload, _ := json.Marshal(body)
        w.Write(payload)
    }
}
```

Criei um Handler para cada rota que vou servir. Alguns são bem simples, como o que lista todos os candidatos:

```
func (h *Handlers) CandidatesHandlerFunc(w http.ResponseWriter, r
*http.Request) {
    Candidates, err := h.Db.Query(`SELECT * FROM Candidates`)
    responseCode := http.StatusOK
    if err != nil {
        fmt.Println(err)
        message := map[string]string{"error": "error connecting to
database"}
        switch err {
        case sql.ErrNoRows:
            responseCode = http.StatusNotFound
            message["cause"] = "no cadidates"
        default:
            responseCode = http.StatusInternalServerError
            message["cause"] = "database error"
        }
        WriteResponse(responseCode, message, w)
        return
    }
    var lista []Candidate
    for Candidates.Next() {
        var Candidate Candidate
        if err := Candidates.Scan(&Candidate.Id, &Candidate.Name,
&Candidate.CreatedAt); err != nil {
            panic(err)
        }
        lista = append(lista, Candidate)
    }
```

```
    WriteResponse(responseCode, lista, w)
}
```

Eu utilizei um "truque" para passar o database para os **handlers**: Coloquei-os como métodos de uma struct, que tem o database como propriedade.

Os outros handlers não são complicados. É só lembrar o que vamos fazer.

Testando

Depois de subir o banco (não precisa criar o database pois estou utilizando o database postgres padrão) e rodar a migration up, é só executar o servidor:

```
go run cmd/main.go
```

(Lembre-se de criar as variáveis de ambiente para ficarem iguais aos seus valores - host, porta, usuário, senha, porta do servidor etc)

Então você pode usar os comandos **cURL** para testar:

```
curl -i http://localhost:8080/candidates
curl -i -X POST http://localhost:8080/candidate -H 'Content-Type:
application/json' -d '{"nome":"New Candidate #4"}'
curl -i -X PUT http://localhost:8080/candidate/4 -H 'Content-Type:
application/json' -d '{"nome":"New Name Candidate #4"}'
curl -i -X DELETE http://localhost:8080/candidate/4 -H 'Content-Type:
application/json'
```