Dataset Pipeline Server

Rafael Campos Nunes

<2019-03-01 sex>

Conteúdo

1 Introdução

Esta aplicação faz parte de uma interface conjunta que é o *dataset pipeline client*, embora esteja em outro repositório. O objetivo desta é prover um serviço através do protocolo HTTP que permita o envio de arquivos e o armazenamento deles em disco local, além de também ter persistência em um banco de dados local (sqlite).

2 O Servidor

O servidor HTTP foi criado utilizando o pacote *net/http* que fornece vários recursos que auxiliam a disponibilização de um. O servidor recebe uma requisição no endereço especificado pelo cliente e de acordo com a requisição retorna ou as páginas e recursos que estão localizadas no diretório *html* ou o resultado da utilização da API REST.

Criei um servidor de rotas utilizando *vanilla* Go com o pacote disponível na biblioteca padrão o que pode ter feito a performance do servidor decair e até mesmo ter vuneralibilidades de segurança. A utilização do gorilla/mux veio a mente somente dias depois em que eu já tinha a estrutura do servidor pronta e mudá-la tomaria um tempo que eu não tinha disponível.

O acesso ao servidor pode ser feita utilizando um browser ou a API REST, disponibilizei duas formas para aumentar a usabilidade da ferramenta.

3 O Banco de Dados

Ao realizar a tarefa responsável por persistir os dados recebidos pelo servidor foi escolhido o tipo de persistência que seria desenvolvida. Uma de duas opções poderia ser adotada:

1. Persistência em SGBD

2. Persistência em memória

A primeira opção foi escolhida e definida que seria um banco de dados relacional fácil de instalar, configurar e utilizar. O SQLite foi escolhido porque se adequa as três características descritas. Pode-se debater também que a utilização de um banco de dados não relacional fosse melhor adequada para compor a solução do problema haja visto que há a necessidade de inserir e acessar uma grande quantidade de dados sem necessariamente ter relação entre os dados do sistema. É um debate onde visualizo a vantagem na utilização do, a titulo de exemplo, MongoDB.

Entretanto não o utilizei pelo fato de já ter montado a estrutura inicial, assim como no servidor, e tinha pouco tempo hábil para mudá-la depois de visualizar a melhor escolha. Além de também não ter em mente esta solução ao início do desenvolvimento, onde se faz claro ser a melhor.

O código que remete ao banco de dados está presente no arquivo *database.go* e contém várias funções de auxílio às operações deste. Inicialização, inserção e seleção facilitam o trabalho com o banco de dados pois abstraem detalhes. Além dessas funções há também uma função específica para capturar o objeto $sql.DB$, com a assinatura $GetHandler(params...string)$, que permite realizar as funções de inserção e seleção no banco de dados.

4 A API

A API REST contém dois recursos, o recurso *get* e o recurso *new*. Os dois recursos podem ser requisitados com um GET ao endereço do servidor utilizando a seguinte sintaxe:

```
1 /v1/get?pk=<XXX>
2 /v1/new?filename=<XXX>&pk=<XXX>&score=<XXX>
```

O servidor é encarregado ou de buscar o valor correspondendo a chave *pk* ou a salvar a nova entrada dada pelos atributos da requisição.

5 Conclusão

Muitas coisas poderiam ser feitas, inclusiva a melhoria em performance utilizando *goroutines*. Entretanto, devido ao tempo hábil pequeno e eu estar me locomovendo em viagens de ônibus (o que me tomou um dia e algumas horas) eu não consegui realizar com o esmero que tinha imaginado.

A utilização do $gorilla/mux$, como citado na seção do servidor, deve ser feita para minar os problemas evidenciados e, além disso, simplificar o código de upload de arquivos para persistir com maior rapidez e sem erros do servidor atestando que muitos arquivos estão abertos - erro que ainda não consegui resolver -