Relatório *Telegram Bot Big Chungus*

Joshua kook ho pereira[1], Luigi Muller Sousa Linhares[1], Tarlison Sander Lima Brito[1], Victor Barbosa Rocha[1]

[1]Discente da disciplina de Sistemas Distribuídos do semestre 2020.1 do curso de Ciência da Computação – Universidade Federal de Roraima (UFRR)

Boa Vista – RR – Brasil

inpincible@hotmail.com, sluigimuller@gmail.com, Britotarlison@gmail.com, victor.barboza.15@gmail.com

Resumo. *Esta atividade tem como finalidade produzir uma aplicação chatbot em Python utilizando RPC com o aplicativo Telegram, obtendo a localização do usuário e armazenando no Banco de Dados e transformar esta aplicação e seus componentes em containers gerenciados pelo Docker.*

O aplicativo **Big Chungus** tem como objetivo proporcionar uma experiência recreativa ao usuário na forma de um jogo no estilo caça ao tesouro. Através do uso de um ***chatbot*** do **Telegram** como moderador, os usuários podem enviar sua localização para tentar encontrar um tesouro.

Toda estrutura do aplicativo é organizada em ***containers*** para cada serviço: banco de dados, os relacionados a redes sociais e o backend. Para o desenvolvimento deste aplicativo foi utilizada a linguagem de programação **python** e suas bibliotecas, juntamente ao **Docker**.

- ## Componentes

1. Micro serviço de comunicação

Para a implementação do projeto, foi utilizada a API de ***chatbots*** do **Telegram** em conjunto com a linguagem de programação **python**, através da biblioteca ***python-telegram-bot****,* como forma de micro serviço de comunicação.

Neste ***container***, tanto a lógica do aplicativo quanto os serviços de redes sociais são implementados, através de gatilhos que disparam com cada comando enviado pelo usuário.

Desta forma, o usuário irá interagir com o ***chatbot*** e através destas interações poderá usufruir das funcionalidades disponibilizadas pela aplicação, as quais estão detalhadas mais adiante.

2. **Micro serviço de banco de dados**

Para o armazenamento de dados da API foi escolhido o banco de dados ***MongoDB***. Para realizar a comunicação entre o banco de dados e a aplicação foi utilizada a biblioteca do python ***pymongo***. A aplicação irá guardar as informações referentes ao usuário e sua respectiva localização, a qual será usada para os demais processos da aplicação.

O banco de dados armazena os **IDs** necessários para a execução do aplicativo (como ID do usuário) e as localizações mais recentes que foram enviadas pelos usuários.

A interface de visualização do banco de dados é inicializada juntamente ao **container** gerado pelo ***docker-compose.***

3. **Micro serviço de interface de visualização**

O micro serviço de interface de visualização do projeto é realizado através da API do **Twitter**, onde as informações são postadas após serem recuperadas do banco de dados e processadas pela aplicação.

Tal funcionalidade foi implementada com o auxílio da biblioteca ***python-twitter.***

- **Funcionamento**

O usuário ao iniciar a partida gera um **id de partida**, esse id é utilizado pelos outros usuários, pois, desta forma quando os tesouros forem enviados e salvo a sua localização no banco de dados é a partir deste **id** que o *core.py* será capaz de realizar a busca no banco de dados e enviar as respostas ao usuário.

Quando o *core.py* realiza as buscas no banco de dados, primeiro verifica o **id** da partida, depois busca se a posição atual do jogador está próxima do tesouro escondido pelo criador da partida (Host) caso ele esteja próximo o *core.py* utiliza a API do twitter e assim faz um *post* no site com o nome do vencedor e o seu **id de partida.**

1. Visão Geral

Para a implementação do projeto como um todo, cada micro serviço foi transformado em um container **Docker** e os **containers** são então ligados e executados em conjunto entre si através do ***docker-compose***

O container da aplicação, que contém o arquivo ***core.py,*** é responsável pela comunicação entre as **APIs** entre diferentes **containers**.

A ideia principal da aplicação é simular um pequeno jogo baseado na localização dos jogadores.

2. Funcionamento do jogo

Existem 3 atores no jogo: o HOST que vai começar o jogo e esconder o tesouro, o(s) JOGADOR(ES) participantes que vão procurar o tesouro e o BOT que vai ser o mediador e vai dizer quem encontrou o tesouro.

O HOST inicia o jogo com a função /comecar_partida.

O BOT vai interpretar e responder com o id_da_partida.

O HOST deve compartilhar esse id_da_partida com os JOGADORES.

Os JOGADORES devem mandar o comando /novo_jogador <id da partida> para o BOT.

Ao receber o comando /novo_jogador <id da partida>, o BOT deve receber o id da partida junto ao comando.

O BOT deve responder se o jogador encontrou a sala e salvar no banco de dados o Nome dos jogadores e um campo de Tentativas (igual a zero).

O HOST deve sair andando pelo ambiente (por exemplo, um parque) para esconder o tesouro.

O BOT vai salvar a localização do HOST como Tesouro e o Horário na coleção cuja chave primária é o id_da_partida.

Os JOGADORES podem sair andando pelo ambiente e usar o comando /marco para receber uma dica do BOT.

O BOT ao receber /marco deve pegar a localização enviada pelo JOGADOR e responder com uma distância em metros, o quão longe ele está do tesouro e adicionar +1 no número de Tentativas do JOGADOR. Caso a localização do JOGADOR é próxima numa margem de erro baixa do tesouro, o jogo termina e o JOGADOR será notificado da vitória.

O JOGADOR e HOST podem enviar /encerrar_partida para o BOT.

O BOT ao receber o comando /encerrar_partida do JOGADOR deve excluir o usuário da partida e ao receber /encerrar_partida do HOST ele deve apagar a partida do HOST e os usuários da partida.

JOGADOR não pode ser HOST ao mesmo tempo e vice versa.

3. Estrutura dos dados

```
partida_collection = {
      “id_host”: id único do HOST no Telegram
      “nome_host”: nome do HOST no Telegram
      “state” :(
            “ESPERANDO_LOCAL”: esperando o HOST enviar a localização,
            “PRONTO”: localização do tesouro armazenado,
            “ENCONTRADO”: tesouro encontrado
            )
      “local_tesouro”: coordenada do tesouro no formato GeoJSON
      “horario_enterro”: horário que foi enterrado
}
jogadores_collection = {
      “id_jogador”: id único do JOGADOR no Telegram
      “nome_jogador”: nome do JOGADOR no Telegram
      “id_partida”: id da partida que o jogador está cadastrado
      “local_jogador”: coordenada do JOGADOR no formato GeoJSON
      “escavacoes”: quantidade de escavações feitas
}
```

Referências

1. <https://api.mongodb.com/python/3.5.1/index.html> Acesso em 05/12/2020
2. <https://github.com/python-telegram-bot/python-telegram-bot> Acesso em 05/12/2020
3. <https://stackabuse.com/accessing-the-twitter-api-with-python/> Acesso em 07/12/2020
4. <https://docs.docker.com/compose/gettingstarted/> Acesso em 09/12/2020
5. <https://docs.docker.com/compose/> Acesso em 11/12/2020
6. <https://stackify.com/docker-build-a-beginners-guide-to-building-docker-images/> Acesso em 11/12/2020
7. <https://docs.docker.com/get-started/part2/> Acesso em 11/12/2020