Universidade Federal de Minas Gerais - DCC

TP2: Tema Livre: Chatbot

Participantes: Guilherme Vieira, Lucas Mundim e Pedro Nascimento

Brasil

12 de junho de 2018

1 Introdução

O trabalho consiste na implementação de um chat bot que acompanha a *Overwatch League*, campeonato de E-sports entre diversos time de diversos países, como Estados Unidos, Coreia do Sul, China e Inglaterra, de Overwatch, jogo da Blizzard. O chat bot tem por objetivo o acompanhamento de times que disputam a *Overwatch League*, sendo que há a possibilidade de ser totalmente personalizado pelo usuário para seguir especificamente o time, ou times, que ele deseja.

1.1 Motivação

Optamos por fazer um chat de bot para a *Overwatch League*, devido ao gosto que temos por e-sports, além disso é algo prático e útil não so para nos, mas como para qualquer um que compartilhe desse interesse na *Overwatch League*.

1.2 Obejtivos

Temos por objetivo o acompanhamento personalizado da *Overwatch League*, permitindo o usuário a customizar sua agenda com quais times deseja seguir e podendo assim manter-se em dia. Além disso, a implementação de conceitos vistos na disciplina de Programação Modular, especificamente alguns Padrões de Projeto e o aprendizado dos mesmos.

2 Implementação

O trabalho foi desenvolvido atráves da integração do nosso programa com duas APIs: a do *BLiP*, plataforma que facilita a comunicação com diversos canais (no caso escolhemos o Facebook Messenger), fornecendo uma interface amigável bem como serviços para a sua utilização; e a API oficial da Blizzard para a *Overwatch League*, a mesma utilizada no site da liga. Para a integração com a segunda, foi utilizado o pacote *REST ease*, que torna toda a utilização de API's REST muito simples, bastando implementar uma interface para o serviço com métodos para cada chamada de API. O fluxo do bot, para o usuário, funciona da seguinte forma: em um *onboarding* inicial o usuário responde qual o time principal que torce e este já é agendado como alertas. Feito isso, é redirecionado para o menu principal com todas as *features* do bot: gerenciamento de alertas, visualização dos próximos jogos dos times marcados para alertas, visualização do site com a agenda completa, visualização das últimas notícias (e do site), visualização dos top5 times da temporada (e completa no site).

Do ponto de vista do desenvolvimento, com relação aos Padrões de Projeto, temos como mais notáveis:

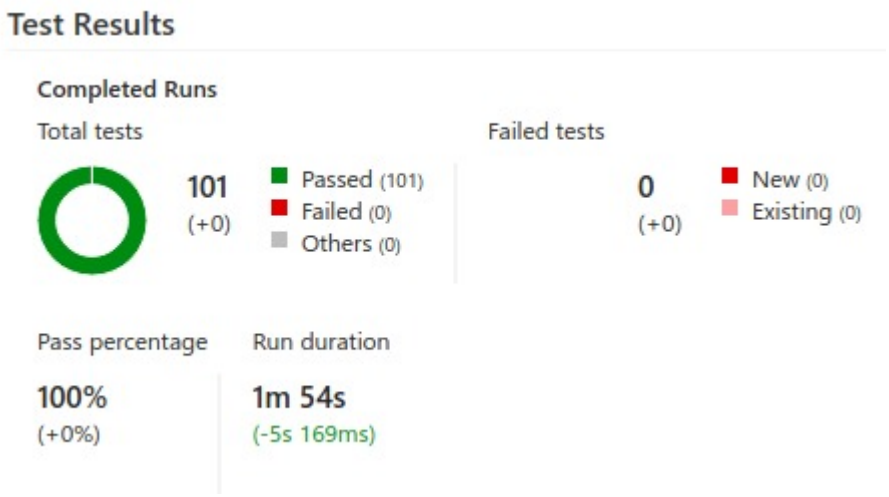
- **Criacionais**
  - **Builder**, utilizado para construir mensagens de tipos complexos, como o carrossel.
  - **Singleton**, todos os serviços são implementados como singletons, via injeção de dependência no *ServiceProvider.cs*. Como não existem métodos explícitos de *GetInstance* nessa implementação, isso não foi possível de ser demonstrado no UML.
- **Comportamentais**
  - **Chain of Responsibility**, na classe *BaseMessageReceiver*, que procura qual (sub)receiver deve receber a mensagem de acordo com as regras estabelecidas no arquivo de configuração *application.json*.
  - **Strategy**, a injeção de dependências no *ServiceProvider.cs* permite que as instâncias de todos os serviços registrados nele sejam acessíveis em qualquer ponto do código.

O fluxo segue a seguinte estrutura, programáticamente: o *BaseMessageReceiver* recebe a mensagem, redireciona para o receiver responsável que chama um serviço de fluxo do

bot para poder fazer toda a manipulação de dados obtidos da API da Blizzard e enviar conteúdo dinâmico para o usuário em forma de conteúdo rico.
De forma mais abstraída, o fluxo por onde a mensagem passa é o seguinte:

2.1 Testes Realizados

Para testar, usamos o *NUnit* e a medição de *code coverage* foi feita pelo *VsTest* do *VisualStudio*.



Resultado dos testes unitários no último *build*.

Cobertura de código proporcionada pelos testes implementados.

2.2 Complementares

- O diagrama UML se encontra em um arquivo juntamente dessa documentação.
- Para utilizar o programa é só executar o comando *make* dentro da pasta TP2 para compilar e *make run* para executar.

- Página para testar o Bot (se ele estiver rodando): <https://www.facebook.com/fireballOWL> ou <https://www.messenger.com/t/fireballOWL> para ir direto ao chat (requer conta do Facebook)
- Página pra testar o Bot (se ele estiver rodando): <https://tinyurl.com/yd9tjfka> (Não requer cadastro algum, cada F5 gera um novo usuário)

Conclusão

A utilização de padrões de projeto é quase uma obrigatoriedade para a produção de qualquer programa mais robusto, especialmente se feito em equipe e se houver planejamento de manutenção em médio ou longo prazo. Ao seguir as linhas guias de um ou mais padrões de projeto, o código torna-se mais legível, limpo, conciso e eficiente, de forma que outros programadores conseguem, com certa facilidade, alterar e/ou corrigir problemas encontrados no programa.

A utilização de padrões de projeto nesse trabalho teve sua contribuição bem nítida: em um programa grande como esse, o trabalho em equipe requer toda a facilidade possível para que tudo se encaixe perfeitamente e que não haja divergências na implementação de cada um.

Além disso, todo o conhecimento adquirido no curso com relação à Programação Modular/Orientada a Objetos foi aplicada, especialmente os princípios *SOLID*, de forma a, novamente, deixar o código reutilizável, inteligível e de fácil manutenção.

Referências

Documentação colaborativa da API da Overwatch League: <https://www.reddit.com/r/Competitiveoverwatch/comments/7p0e8d/owl_api_analysis/>

Documentação oficial do BLiP: <https://docs.blip.ai>

Site oficial da Overwatch League: <https://overwatchleague.com>