UNIVERSIDADE FEDERAL DE MINAS GERAIS
PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II

# Trabalho Prático

Documentação - Twitter

*Arthur Melo*
*André Carmo*
*Fabyo SIlveira*
*Igor Oliveira*

# Sumário

# 1 Introdução

Neste trabalho foi implementado, em **C++**, um sistema que simula uma rede social estilo twitter, onde se pode postar tweets, que são pequenos textos, repostar esses tweets e curtí-los. Há também a funcionalidade de conversas, onde é possível criar uma conversa com outro usuário e mandar mensagens. Além disso, é possível buscar e visualizar perfis de outros usuários, podendo seguí-los e ver seus dados.

No restante da documentação serão apresentadas as principais classesdo projeto e algumas decisões de projeto. Não será necessário explicar cada um dos métodos implementados.

Para executar o projeto basta executar o comando abaixo na pasta raiz.

```
make
make run (Linux)
make run_windows (Windows)
```

Para limpar os arquivos de build, basta executar o comando:

```
make clean (Linux)
make clean_windows (Windows)
```

Para executar os testes unitários implementados basta executar o comando:

```
make test
make runtest (Somente Linux)
```

# 2 Implementação

O projeto foi implementado e modularizado tendo como base a estrutura de Model, View e Controller(MVC). Onde os Models são as representações das tabelas do banco de dados, os Views são as classes de interação com o usário, as interfaces, e os Controllers são as classes que manipulam os dados e fazem a "ponte"entre o banco de dados e as interfaces.

Para o armazenamento dos dados, foi utilizado o banco de dados SQLite, nos próximos tópicos será mostrado como ele foi implementado.

## 2.1 As Views

Essas são as classes responsáveis pela interação com o usário, mostram todos os menus do sistema, solicitam os dados dos usuários, e enviam para os Models/Controllers fazerem os devidos tratamentos:

- *MainScreen*
- *MessageScreen*
- *PostScreen*
- *SearchScreen*
- *SystemScreen*
- *UserScreen*

A classe *MainScreen* mostra o menu principal para o usuário e possui um método virtual chamado *showMenu()* que faz com que todas as outras views(*PostScreen, MessageScreen, etc*) implementem esse método. Assim foi criado uma herança entre as views específicas e a *MainScreen* possibilitando que esta controle toda a interface do sistema. As outras views mostram seus respectivos menus.

## 2.2 Os Models

Essas classes representão as tabelas do sistema e são responsáveis por receber os dados das views e tratá-los juntamente com os DAOs.

- *Post*
- *Message*
- *User*
- *Search*
- *System*

A classe *System* incializa o sistema, criando as tabelas na primeira execução e armazenando o usário logado, para fornecê-lo para o resto do sistema. A classe *Search*, é responsável pelas buscas de *User* e *Post*. Os outros Models representam as tabelas do SQLite.

## 2.3 Os DAO's

Essas classes são as responsáveis por comunicar com o banco. Essas fazem as queries e as transformam em Models, para que as views possam manipular os resultados mais facilmente

- *DAO*
- *Connection*
- *SystemDAO*
- *PostDAO*
- *MessageDAO*
- *SearchDAO*
- *UserDAO*

Aqui é onde são feitas praticamente todas as manipulações dos dados. Nesta parte foi criada uma hierarquia onde a classe *DAO* é pai de todas as outras, exceto a *Connection* que é a representação da conexão com o SQLite. Na classe DAO foram implementados todos os métodos básicos de comunicação com o banco (create, insert, update, delete, etc), ela é a responsável por receber as queries dos DAOs filhos e retornar os dados processados pela *Connection*.

### 2.4 Singletons

Um conceito interessante utilizado nesse trabalho foram os Singletons. Ao decidir trabalhar com o SQLite para o armazenamento de dados, nos deparamos com um problema: as múltiplas instâncias da classe *Connection*, que era instanciada cada vez que um dos DAOs eram inicializados, essas múltiplas instâncias geravam uma série de inconsistências no banco de dados.

Após pesquisas, encontramos o conceito de Singleton, que é um modo de implementar uma classe, onde ela nunca é instanciada mais de uma vez. Essa implementação foi feita através de construtor privado e métodos estáticos.

Como o Singleton na classe *Connection* funcionou muito bem, decidimos aplicá-lo também na classe *System* que mantém só uma instância do sistema, armazenando dados úteis para as outras classes.

## 3 Conclusão

Uma das maiores dificuldades que encontramos no desenvolvimento do trabalho foi a modelagem inicial. Apesar dos Users Stories e dos Cartões CRC's nos dar uma primeira visão de como seria o sitema, com o passar do tempo, e das linhas de códgio, percebemos que a modelagem não seria a melhor e tivemos que mudá-la, diversas vezes, até encontrar uma estrutura, onde as classes e pastas ajudassem no desenvolvimento das funcionalidades.

No mais, o trabalho foi de grande ajuda para pôr em prática os conceitos vistos na aula(Herança, Polimorfismo, Composição e etc).

## 4 Referência


- SQLite Tutorial
- SQLite Códigos de erro
- Singleton