# Projeto Prático

## 1 Introdução

O grupo deve escolher um problema de seu interesse e realizar todo o processo de desenvolvimento (análise, projeto e implementação) de uma solução. Espera-se, ao final, um sistema de porte médio, que utiliza os conceitos e técnicas vistos durante o curso (modelagem, conceitos de POO, testes unitários, boas práticas, etc). O programa deve ser feito baseado na linguagem C++17. Uma lista não exaustiva de *sugestões* de temas é apresentada abaixo. O tema é aberto à negociação e os alunos também podem sugerir outros temas de seu interesse.

- Twitter (tweet, retweet, follow, etc)
- Gerenciador de tarefas/compromissos (adicionar, remover, listar, histórico, etc)
- Sistema para biblioteca (busca, reserva, empréstimo, etc)
- Sistema de gerência de e-commerce
- Mini rede social (timeline, adicionar/listar amigos, post, etc)
- UFMG: Carona, Eventos, Bandejão, Gamefication, Oportunidades, …
- Biblioteca de grafos
- Indexador/organizador de arquivos
- Outros (Magic, RPG, Reserva de Passagens/Hotel, Netflix, Spotify, Trello/Notion, …)

**ATENÇÃO: Temas similares poderão ser escolhidos por no máximo 3 grupos!**

O desenvolvimento e a entrega deverão ser feitos utilizando o sistema de controle de versão GitHub. Sugere-se que commits/pushs sejam feitos de maneira frequente, por todos os membros do grupo, sempre que houver alterações.

Lembre-se, o objetivo não é apenas escrever um programa funcional, mas desenvolver um sistema confiável, reutilizável e de fácil manutenção e extensão! Logo, tente aplicar todos os conceitos de POO, modularidade e corretude vistos em sala de aula. Também serão avaliados critérios como criatividade na solução, assim como a possível implementação de funcionalidades extras

Durante o desenvolvimento, o grupo deverá utilizar o framework doctest (`https://github.com/onqtam/doctest`) para implementar os testes de unidade (o mesmo utilizado na sala de aula). Deve haver pelo menos uma classe de testes para cada uma das principais classes do sistema (por exemplo, as classes de entidades).

A interface de interação com a aplicação poderá ser feita via terminal de comando. (A implementação de aplicações gráficas entra como ponto extra, mas não se sinta pressionado a fazer, tendo em vista que não faz parte da ementa da disciplina.) Possíveis arquivos necessários durante a etapa de inicialização do sistema deverão ser fornecidos pelo grupo.

## 2 Modelagem

As histórias de usuário são uma forma simples de apresentar os requisitos funcionais desejados para um determinado sistema. São artefatos de desenvolvimento utilizados principalmente em processos baseados em metodologias ágeis. As descrições são intencionalmente genéricas, dando liberdade ao grupo para decidir detalhes da implementação.

Nesta etapa, o grupo deverá identificar possíveis funcionalidades interessantes de serem incorporadas ao sistema e propor pelo menos CINCO histórias de usuário. Para cada uma, deverá ser feita uma descrição sucinta. Tente apresentar histórias de usuário para diferentes tipos de

usuário do sistema (ou considerando papeis específicos em certos contextos). Cada história de usuário deve apresentar entre três e cinco critérios de aceitação.

**Entrega**: o grupo deverá utilizar a ferramenta de *issues* do GitHub para criação e controle das histórias de usuário. Cada história corresponderá a uma issue. O título da issue será a história de usuário, e o corpo da issue deve conter a lista de critérios de aceitação como checkboxes. Critérios de aceitação muito complexos talvez devem se tornar uma história por si só (a depender do julgamento do grupo); é possível converter uma tarefa da lista pelo próprio GitHub, passando o mouse sobre o item da lista.

Em seguida, faça pelo menos CINCO cartões CRC para as classes que você julga como sendo as mais importantes. Cada cartão deve apresentar pelo menos quatro responsabilidades *coesas* (considerando conhecimento e realização) e mencionar pelo menos duas colaborações. Você pode usar esse site para lhe auxiliar a fazer o cartão: https://echeung.me/crcmaker/

# 3 Documentação

Durante toda a implementação, quaisquer declarações de classes e funções (apenas nos arquivos de cabeçalho!) devem ser documentadas utilizando-se a ferramenta Doxygen (http://www.doxygen.org/). A sintaxe dos comentários pode ser encontrada nas notas de aula (os comentários com três barras, seguido de notações especiais como @brief, @param, e @return).

Além disso, o grupo deve utilizar o arquivo README.md (https://docs.github.com/pt/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes) do GitHub para documentar o projeto em si. O arquivo README.md deve conter (pelo menos!):

- Uma breve apresentação do projeto/problema
- Menções a quaisquer dependências que sejam necessárias para o correto funcionamento
- Instruções de compilação e execução (incluindo, caso aplicável, a necessidade de um sistema operacional específico; se possível, para facilitar a correção, peço que garantam o funcionamento no ambiente linux/wsl)

# 4 Comentários Gerais

- O trabalho deverá ser feito em grupos com quatro ou cinco alunos
- Trabalhos copiados serão, obviamente, zerados!!!
- Na entrega final, será considerado o último commit na branch principal do projeto
- O arquivo deve conter um arquivo Makefile com as opções make e make run

# 5 Critérios de Avaliação

## 5.1 Entrega Parcial (Modelagem)

### 5.1.1 Histórias de Usuário (3 pontos):

- $-0.6$ pontos por cada história não entregue (considerando o mínimo de cinco)
- $-0.1$ ponto por cada critério de aceitação não entregue, incoerente com a história, ou superficial (mínimo três por história)
- $-0.3$ pontos por cada história muito simples / pouco expressiva
- Em caso de mais de cinco histórias, a nota corresponderá as cinco mais bem availadas

#### 5.1.2 Cartões CRC (3 pontos):

- $-0.6$ pontos por cada cartão CRC não entregue (considerando o mínimo de cinco)
- $-0.1$ ponto por cada responsabilidade não entregue (mínimo quatro por cartão)
- $-0.1$ ponto por cada colaboração não entregue (mínimo dois por cartão)
- Em caso de mais de cinco cartões, a nota corresponderá aos cinco mais bem avaliados

### 5.2 Entrega Final (Implementação)

#### 5.2.1 Documentação e Estilo (4 pontos):

- $-2$ pontos se README incompleto/pouco detalhado
- $-2$ pontos se não utiizou/utilizou incorretamente o Doxygen
- $-1$ ponto se descrição das classes/funções pouco detalhadas
- $-1$ ponto se nomes de atributos e funções não padronizados
- $-1$ ponto se indentação não padronizada

#### 5.2.2 Funcionamento (6 pontos):

- $-6$ pontos se sequer compila
- $-4$ pontos se compila, mas não executa
- $-0.5 \sim 1$ ponto por cada erro durante a execução, a depender do erro

#### 5.2.3 Boas Práticas e POO (6 pontos):

- $-1$ pontos se não utilizou o conceito de encapsulamento corretamente
- $-1$ pontos se não utilizou composição/herança/interfaces corretamente
- $-1$ pontos se não utilizou polimorfismo corretamente
- $-1$ ponto se não modularizou o código (arquivos hpp e cpp)
- $-1$ ponto se não modularizou o projeto (diferentes diretórios)
- $-2$ pontos se não criou o Makefile

#### 5.2.4 Programação Defensiva / Tratamento de Exceções (4 pontos)

- $-4$ pontos se não fez nenhum tratamento/sanitização das entradas
- $-2$ pontos se não fez tratamento de exceções corretamente

#### 5.2.5 Testes de Unidade (4 pontos)

- $-4$ pontos se não fez nenhum teste de unidade
- $-2$ pontos se faltaram testes para classes importantes/principais

#### 5.2.6 Criatividade (2 pontos extras)

### 5.3 Nota Final

Será avaliada a partipação individual de cada membro do grupo, como um valor entre 0 e 1, baseado na proporção de commits no GitHub e interação durante as aulas de acompanhemento do projeto, por email, mensagens, etc.