Documento de Arquitetura de Software

| **Grupo** |
| --- |
| Gabriel Gomes |
| Ramon Pedro |
| Pedro Umpierre |
| Lucas Vitorino |
| Frederico Antunes |
| Willian Benevides |

| **Objetivo deste Documento** |
| --- |
| Este documento tem como objetivo descrever as principais decisões de projeto tomadas pela equipe de desenvolvimento e os critérios considerados durante a tomada destas decisões. Suas informações incluem aparte de *hardware* e *software* do sistema. |

**Sumário**

[1.](#_heading=h.c7b8hnp79xey) INTRODUÇÃO 2

[1.1](#_heading=h.te6w57tiq1r0) Finalidade 2

[1.2](#_heading=h.sl5desqjvlrx) Escopo 2

[1.3](#_heading=h.2vvyv8k6jap9) Levantando funcionalidades 3

[2.](#_heading=h.u1anf5u6d8qp) DESCRIÇÃO DOS OBJETIVOS, RESTRIÇÕES E REQUISITOS ARQUITETURAIS 4

[3.](#_heading=h.y7zgk4w7zzwt) JUSTIFICATIVA 6

4. DIAGRAMA DE CASOS DE USO 7

5. MODELO CONCEITUAL 8

6. DIAGRAMA DO PADRÃO ARQUITETURAL 9

7. DIAGRAMA DE CLASSES

8. DIAGRAMAS DE SEQUÊNCIA

# INTRODUÇÃO

## Finalidade

Este documento fornece uma visão arquitetural abrangente do sistema Beluken - RPG Battle Manager, usando diversas visões de arquitetura para **representar** diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

## Escopo

O sistema proposto é um aplicativo multiplataforma, desenvolvido em Flutter, com suporte para os sistemas operacionais Windows e Android. Seu objetivo principal é oferecer uma ferramenta completa para a criação, gerenciamento e simulação de batalhas em cenários de RPG. Inspirado em sistemas tradicionais de jogos de interpretação de papéis, o aplicativo se propõe a apresentar um sistema próprio de combate, com foco na fluidez e dinamismo das interações durante os confrontos.

O aplicativo permitirá ao usuário criar fichas detalhadas tanto de personagens quanto de inimigos. Essas fichas contém informações como raça, classe, arquétipo, nível inicial, atributos distribuídos, equipamentos e uma descrição geral. No caso dos inimigos, será possível definir também o tipo, além dos outros dados mencionados. Essas informações servirão como base para as batalhas e para o gerenciamento dos grupos que participarão delas.

O sistema fornecerá funcionalidades específicas para o gerenciamento de personagens e inimigos, permitindo ao usuário editar, duplicar ou excluir fichas já criadas. Com base nesses cadastros, será possível criar grupos de personagens com até quatro membros, e grupos de inimigos, que poderão conter um ou mais monstros. Esses grupos poderão ser salvos, modificados ou reutilizados conforme o interesse do usuário.

Para iniciar uma batalha, o usuário deverá configurar uma nova instância de combate, selecionando um grupo de personagens, um grupo de inimigos e um cenário previamente definido. Esses passos fazem parte do processo de configuração e são essenciais para que o combate seja iniciado de forma adequada.

Durante a batalha, o sistema assumirá o papel de gerenciador do combate, permitindo ao usuário conduzir o andamento da luta por meio do registro das ações realizadas por cada personagem ou inimigo. O sistema acompanhará turnos, calculará efeitos de habilidades, atualizará atributos como pontos de vida e mana, e apresentará feedbacks visuais e textuais que reflitam o estado atual do combate. O histórico das ações será mantido para fins de acompanhamento e tomada de decisões táticas.

Todas as informações geradas, fichas, grupos, cenários e batalhas realizadas serão armazenadas localmente em banco de dados, garantindo a persistência dos dados entre sessões de uso. A escolha da base de dados será compatível com o Flutter, sendo recomendadas soluções como SQLite.

O aplicativo será desenvolvido com uma interface amigável e responsiva, adequada tanto ao uso com mouse e teclado no Windows quanto ao toque em dispositivos Android. A navegação entre as seções será intuitiva, facilitando o acesso às funcionalidades de criação, configuração e gerenciamento.

Embora o sistema de combate seja baseado em estruturas tradicionais de RPG, ele contará com regras e mecânicas próprias, pensadas para tornar as batalhas mais rápidas e acessíveis. Essas regras poderão ser estendidas futuramente, incluindo variações, personalizações ou modos alternativos. O sistema é projetado com flexibilidade para permitir futuras extensões de funcionalidades e regras dentro do ambiente local.

## Levantando Funcionalidades:

O sistema proposto é um aplicativo desenvolvido em Flutter para Windows e Android com a função de gerenciar batalhas entre grupos de fichas de personagens e grupos de monstros. No aplicativo o usuário deve ser capaz de:

* Criar e gerenciar grupos de fichas de personagens que podem variar de 1 membro até 4 membros.
* Criar fichas de personagens para serem colocados nos grupos.
* Criar fichas de monstros.
* Disponibilizar alta customização estilo RPG para fichas de personagens na hora da criação e posteriormente.
* Disponibilizar alta customização estilo RPG para fichas de monstros na hora da criação e posteriormente.
* Manter os grupos salvos em um banco de dados local.
* Manter as fichas de personagens salvas em um banco de dados local.
* Manter as fichas de monstros salvas em um banco de dados local.
* Adicionar e manter cenários no banco de dados local.
* Permitir a construção de batalhas ao selecionar grupos de fichas de personagens, 1 ou mais monstros e um cenário.
* Gerenciar as batalhas.

1. [**DESCRIÇÃO DOS OBJETIVOS, RESTRIÇÕES E REQUISITOS ARQUITETURAIS**](#_heading=h.u1anf5u6d8qp)

Esta seção delineia as metas de qualidade que a arquitetura visa alcançar (Objetivos), as condições e decisões preexistentes que devem ser respeitadas (Restrições), e as capacidades específicas que a arquitetura deve prover para atender às necessidades do sistema (Requisitos).

* **Restrições Arquiteturais**
  + **Restrição 1: Suporte Multiplataforma com Código Reutilizável**
    - **Descrição:** O projeto exige a entrega de uma aplicação funcional em, no mínimo, duas plataformas distintas (Windows e Android) a partir de um esforço de desenvolvimento unificado. É uma premissa do projeto otimizar os recursos de desenvolvimento e garantir a consistência da experiência do usuário entre as plataformas, evitando a criação e manutenção de bases de código separadas.
    - **Decisão Arquitetural Resultante:** Para atender a esta restrição de eficiência e portabilidade, **foi decidido** utilizar o framework Flutter, que permite o desenvolvimento a partir de uma única base de código em Dart.
  + **Restrição 2: Operação Independente de Conectividade**
    - **Descrição:** O sistema deve ser totalmente funcional em cenários sem acesso à internet. O caso de uso principal envolve o gerenciamento de sessões de RPG de mesa, que frequentemente ocorrem em ambientes onde a conectividade pode ser instável ou inexistente. A capacidade de operar offline é, portanto, um requisito operacional inegociável.
    - **Decisão Arquitetural Resultante:** Para garantir a funcionalidade offline, **foi decidido** que todo o armazenamento de dados (fichas, grupos, batalhas) será estritamente local, no dispositivo do usuário, sem dependência de um backend ou serviços na nuvem.
  + **Restrição 3: Escopo de Lançamento Focado**
    - **Descrição:** Para viabilizar a entrega e validação do produto dentro do cronograma do projeto, o escopo da versão inicial está focado e limitado. A prioridade é validar as funcionalidades centrais em um conjunto definido de ambientes antes de considerar uma expansão mais ampla.
    - **Decisão Arquitetural Resultante:** Em função desta restrição de escopo, **foi decidido** que o lançamento inicial e os testes se concentrarão nas plataformas Windows (desktop) e Android (móvel).
* **Objetivos Arquiteturais:**
  + **Manutenibilidade e Extensibilidade:** Alcançar uma arquitetura de software que seja **altamente fácil de manter e estender**. Isso significa que modificações, correções de bugs e a adição de novas funcionalidades locais (como novas regras de RPG, tipos de personagens ou mecânicas de combate) devem ser possíveis com impacto mínimo nos componentes existentes.
  + **Testabilidade:** Garantir um **alto grau de testabilidade** para todos os componentes críticos do sistema. A arquitetura deve facilitar a criação de testes unitários para a lógica de negócios (regras de combate, criação de fichas) e testes de integração que validem as interações entre as camadas.
  + **Flexibilidade para Evolução Local:** Prover **flexibilidade arquitetural** para permitir a evolução do sistema em termos de regras de jogo, personalizações e modos de operação, mantendo o foco em funcionalidades que operem localmente no dispositivo do usuário.
  + **Desempenho Responsivo:** Assegurar que o sistema apresente **baixo tempo de resposta e fluidez** nas interações do usuário, especialmente durante operações críticas como a simulação de batalhas, cálculos de atributos e navegação entre telas.
  + **Usabilidade e Experiência do Usuário:** Facilitar, através de uma estrutura bem definida, a construção de uma interface de usuário que seja **intuitiva, agradável e eficiente**, adaptando-se adequadamente a diferentes contextos de uso (desktop e mobile).
* **Requisitos Arquiteturais:**
  + **Interface Adaptativa:** O sistema **deve** suportar layouts de interface que se adaptem de forma responsiva a diferentes tamanhos de tela, densidades de pixel e orientações (retrato/paisagem), visando uma experiência consistente em dispositivos desktop (Windows) e móveis (Android), e potencialmente web e iOS no futuro.
  + **Gerenciamento Eficiente de Dados Locais:** A arquitetura **deve** ser capaz de gerenciar eficientemente um volume crescente de dados do usuário (fichas de personagens e inimigos, grupos, cenários, históricos de batalha) armazenados localmente em um banco de dados SQLite, garantindo a integridade e a rápida recuperação dessas informações.
  + **Segurança dos Dados Locais (Opcional):** O sistema **deve** permitir a futura integração de mecanismos de criptografia para os dados armazenados localmente, caso a proteção da privacidade das informações do usuário contra acesso não autorizado no dispositivo seja identificada como uma necessidade.
  + **Navegação Clara e Consistente:** A arquitetura **deve** suportar um sistema de navegação claro, previsível e guiado, permitindo que os usuários acessem todas as funcionalidades principais com facilidade.
  + **Suporte a Temas e Diretrizes de Design:** O sistema **deve** permitir a aplicação de temas visuais (como claro e escuro) e, na construção dos componentes de interface, considerar as diretrizes de design Material Design (para Android, Windows, Web) e Cupertino (para iOS) para uma integração visual adequada às respectivas plataformas.
  + **Aplicação dos Padrões MVC e Repository:** A estrutura da aplicação **deve** ser organizada seguindo o padrão Model-View-Controller (MVC) para a separação de responsabilidades entre apresentação, lógica de controle e dados. Adicionalmente, **deve** ser utilizado o padrão Repository para abstrair e gerenciar o acesso à camada de persistência de dados local.

# JUSTIFICATIVA

**Manutenibilidade:** A adoção do padrão MVC (Model-View-Controller) permite a separação clara entre a lógica de negócios, a interface do usuário e o controle da aplicação. Essa separação facilita a localização de erros, a modificação de funcionalidades existentes e a introdução de novas funcionalidades, uma vez que as camadas estão desacopladas. Isso reduz a complexidade do código e melhora a legibilidade e organização do sistema. Além disso, o uso do padrão Repository contribui significativamente para a manutenibilidade ao desacoplar a lógica de acesso a dados do restante da aplicação. Isso permite que o mecanismo de persistência local (SQLite) seja gerenciado ou até mesmo reestruturado internamente sem impactar diretamente outras partes do sistema que consomem os dados.

**Capacidade de Crescimento Local (Escalabilidade):** O sistema foi concebido com foco na capacidade de lidar com um aumento no volume de dados gerados pelo usuário (fichas, grupos, históricos de batalha) de forma eficiente no ambiente local. O uso de Flutter com código base único permite escalar a aplicação para diferentes plataformas (Android, iOS, Windows e web), garantindo consistência funcional e estética em todas elas, caso o escopo de plataformas seja expandido no futuro. A arquitetura modular também visa facilitar a adição de novas funcionalidades locais.

**Portabilidade:** A escolha pelo framework Flutter com Dart permite a compilação do mesmo código-fonte para múltiplas plataformas, facilitando a manutenção de versões consistentes do aplicativo e reduzindo custos de desenvolvimento. Isso garante portabilidade entre dispositivos móveis, desktop e navegadores modernos, atendendo à restrição de compatibilidade com Android, iOS, Windows, Chrome, Firefox e Safari.

**Testabilidade:** A estruturação do sistema segundo os princípios do MVC e o uso do padrão Repository (com injeção de dependência) favorecem a criação de testes automatizados, tanto unitários quanto de integração. Cada componente da lógica de negócios pode ser testado isoladamente da camada de persistência, mockando-se as interfaces do Repository, garantindo que as regras de negócio, cálculos de combate e manipulação de dados estejam corretos.

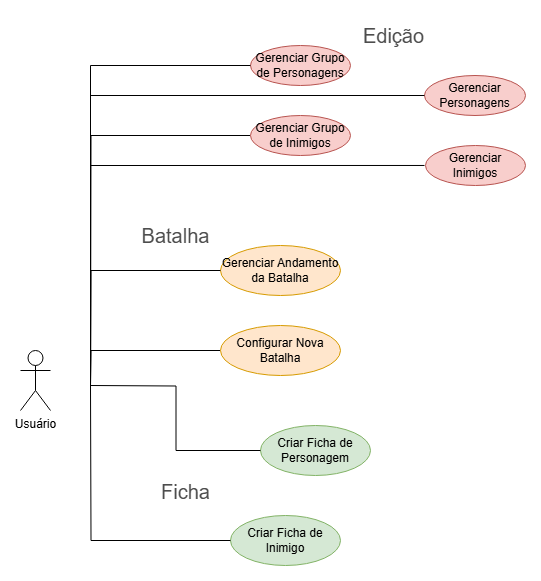
**Segurança:** O sistema armazenará todos os dados localmente. Caso seja necessário, a arquitetura permite a integração de mecanismos de criptografia para proteger os dados do usuário armazenados no dispositivo.

**Usabilidade:** A arquitetura propõe o uso de layouts adaptativos e navegação guiada com suporte a temas claro e escuro, o que contribui para uma experiência de usuário consistente, intuitiva e acessível em diferentes dispositivos. A divisão clara das responsabilidades no MVC facilita também a construção de interfaces responsivas e bem organizadas.

**Desempenho:** O uso de banco de dados local leve, como SQLite, e o gerenciamento local de estados e dados garantem tempo de resposta baixo para operações críticas como atualização de atributos durante as batalhas. O padrão Repository pode ser implementado de forma eficiente para não introduzir sobrecarga perceptível, enquanto organiza o acesso aos dados.

**Flexibilidade e Evolução:** O uso do padrão Repository e a organização modular baseada em MVC garantem que o sistema possa ser expandido futuramente com novas funcionalidades *locais*. O Repository, em particular, oferece flexibilidade para adaptar ou evoluir o mecanismo de persistência de dados (atualmente SQLite) ou introduzir novas formas de gerenciamento de dados *localmente*, sem exigir reestruturações profundas na lógica de negócios.

# DIAGRAMA DE CASOS DE USO

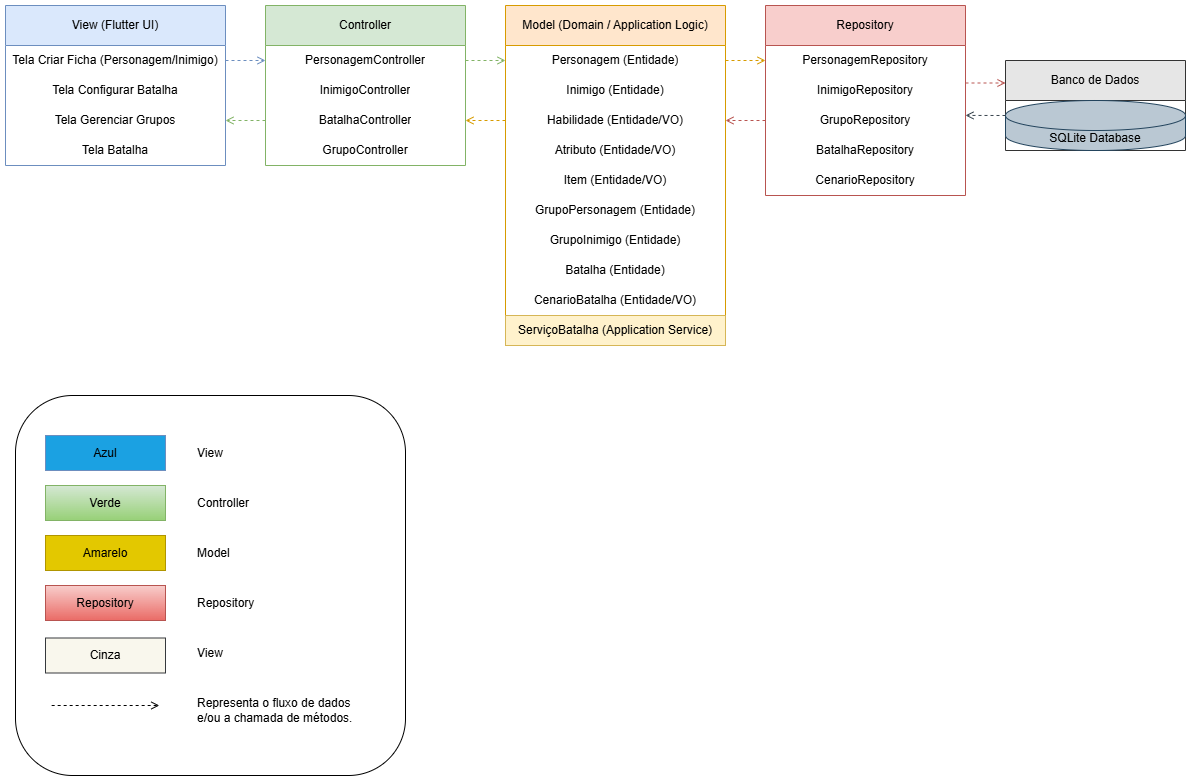
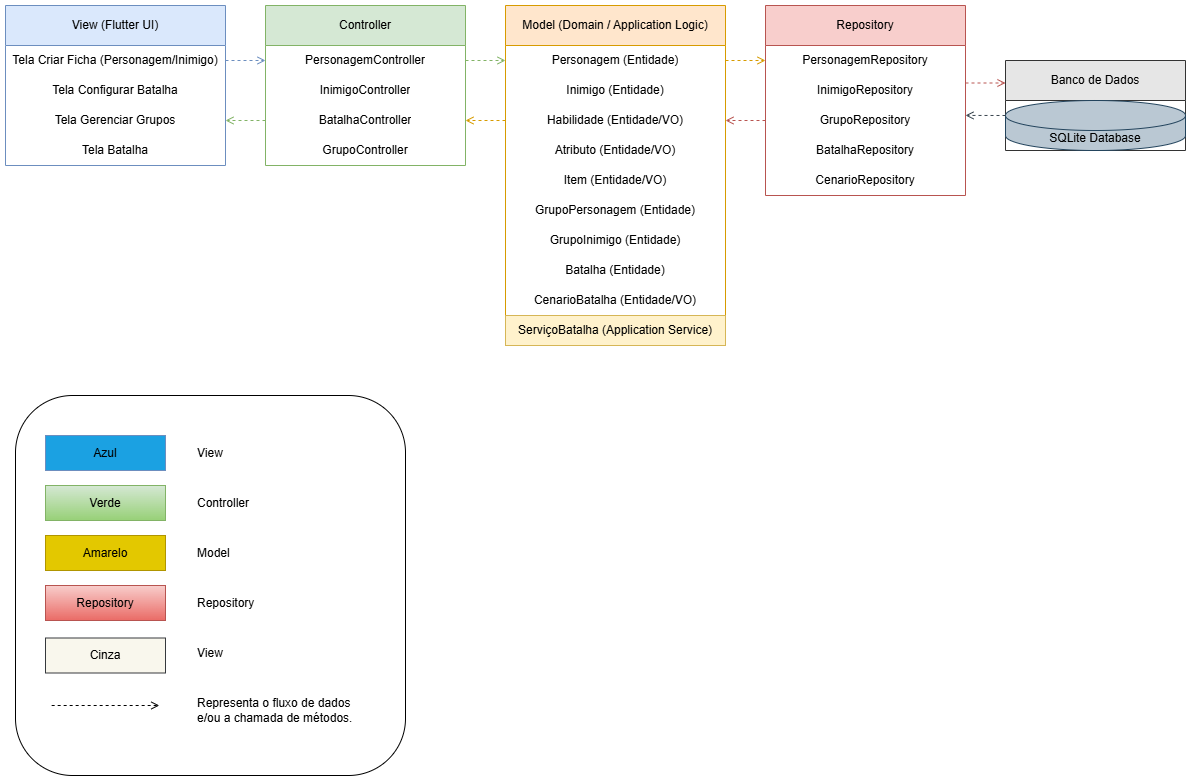


*Figura 3 – Exemplo de Diagrama de Pacotes da Aplicação*

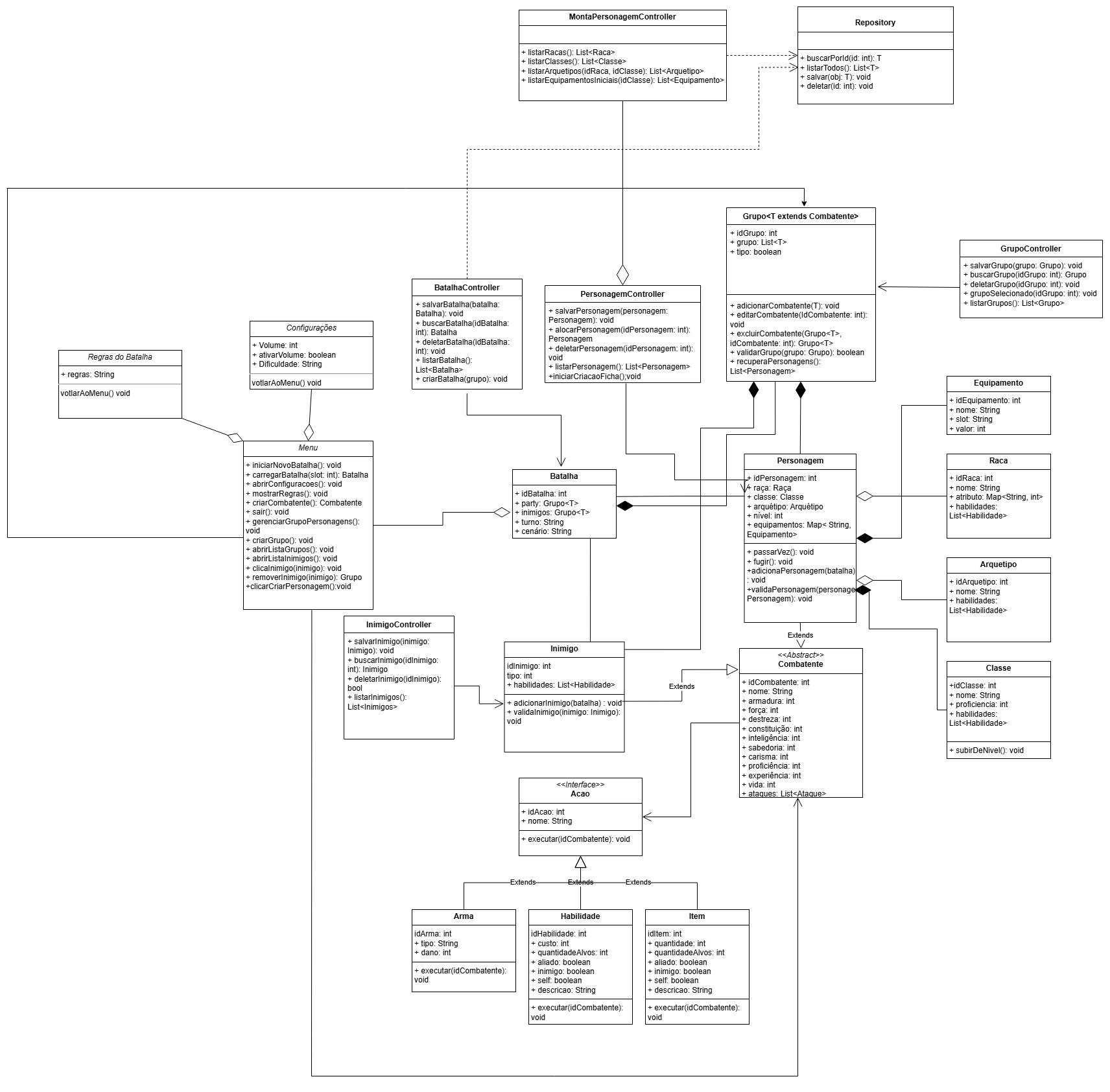
# MODELO CONCEITUAL

# *Modelo Conceitual.drawio.png*

# DIAGRAMA DO PADRÃO ARQUITETURAL

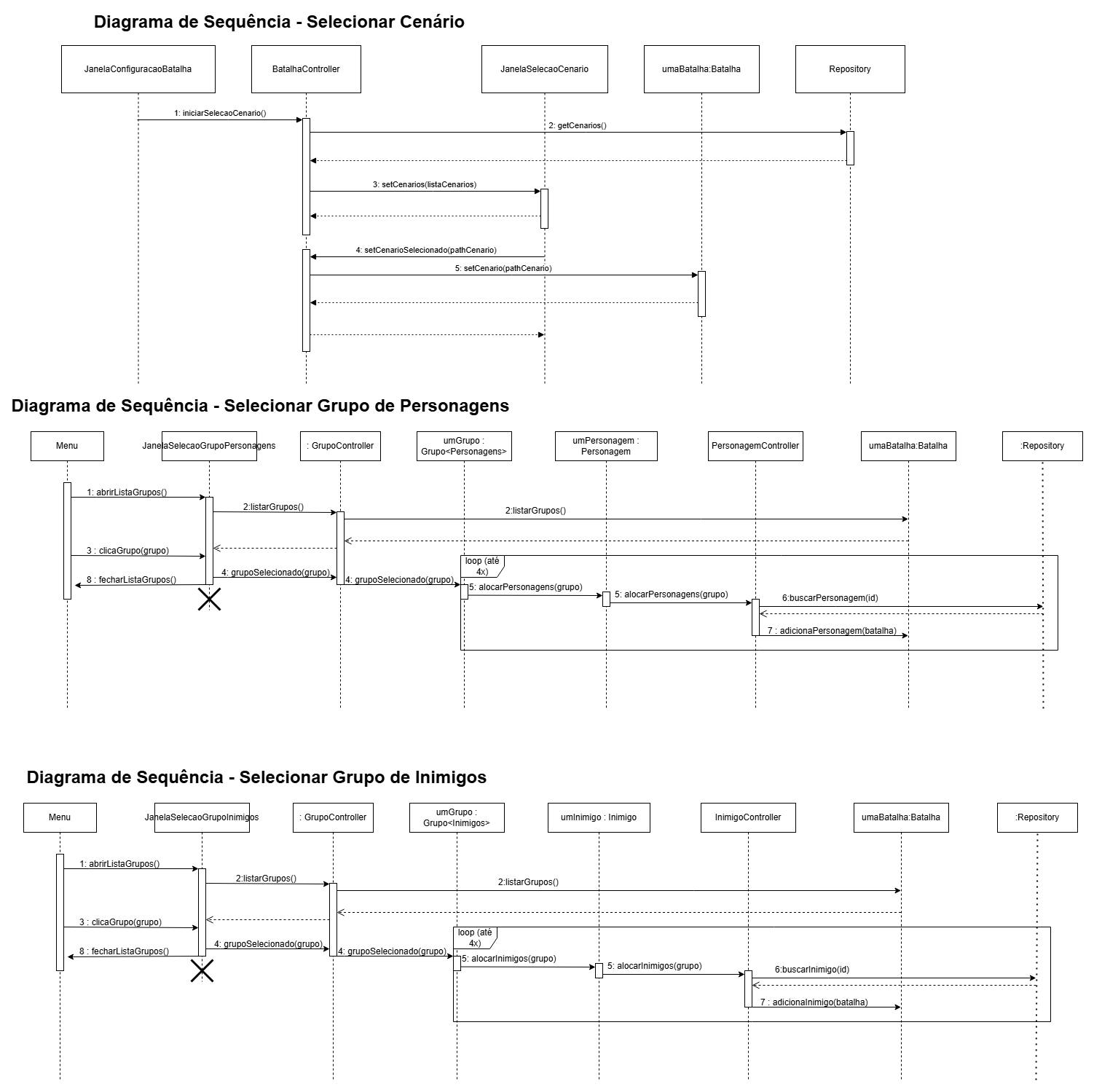
  


# DIAGRAMA DE CLASSES

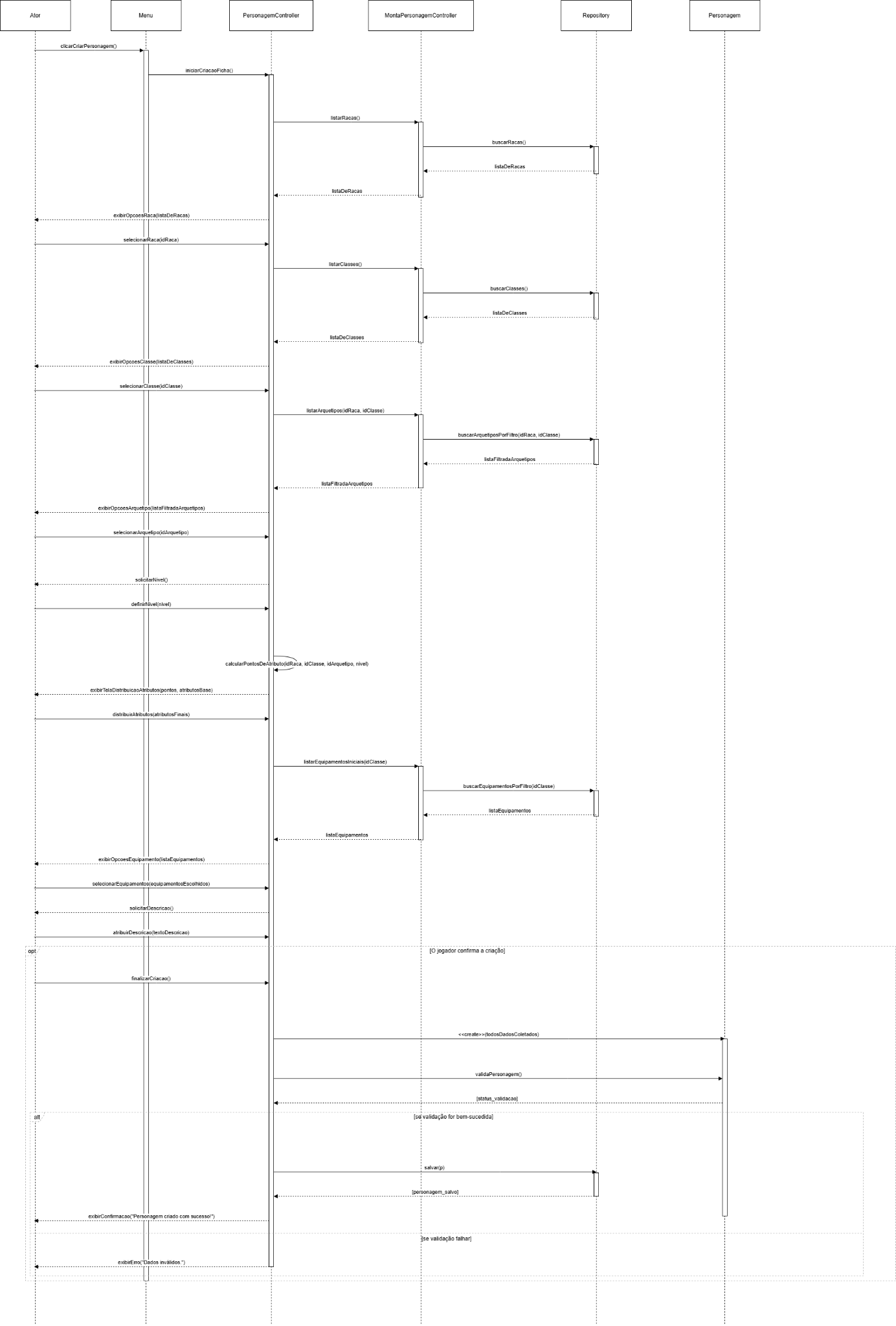


# DIAGRAMAS DE SEQUÊNCIA

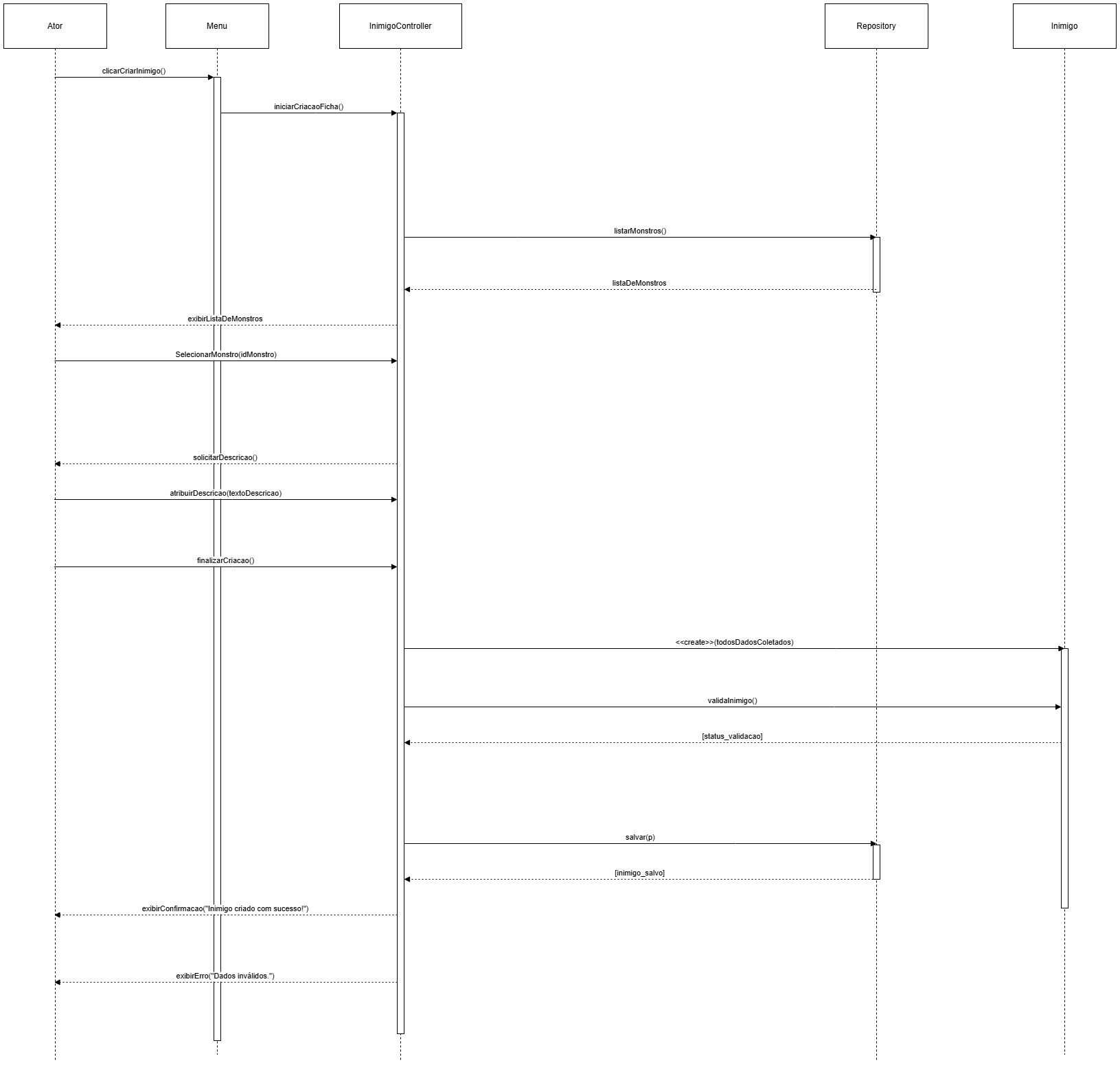
Configurar Nova Batalha



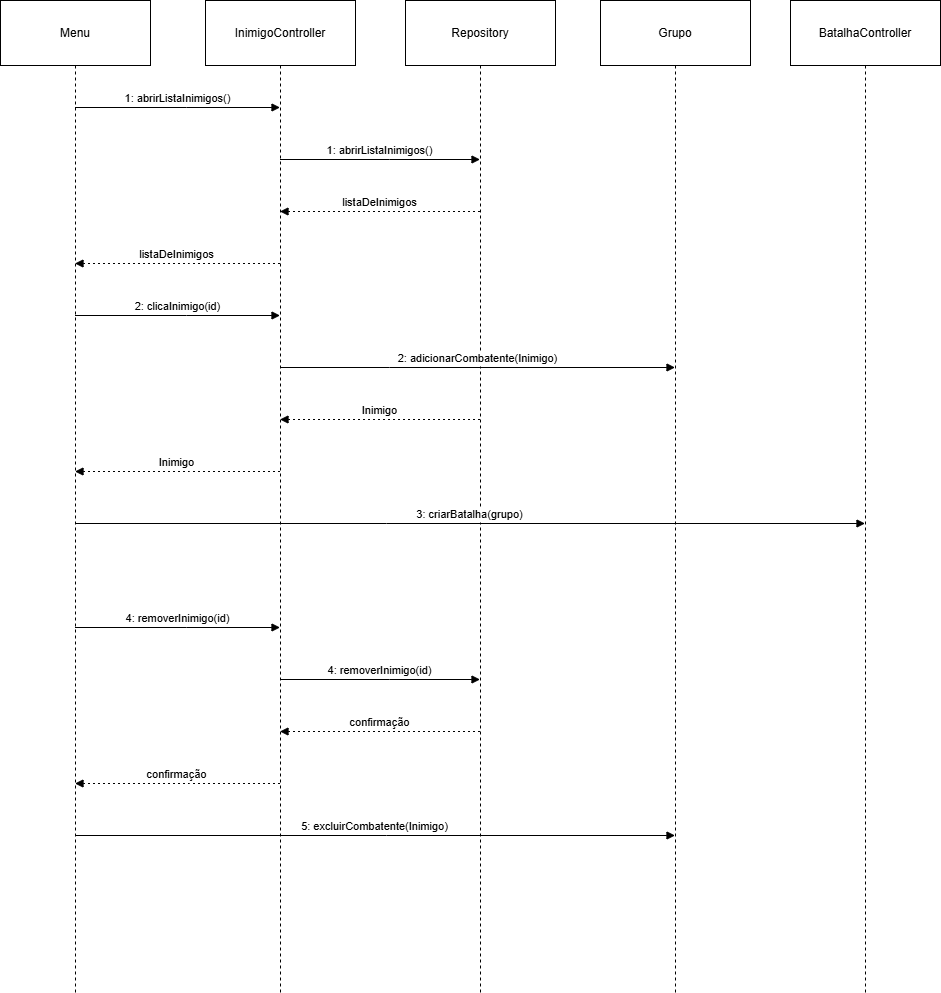
Criar Ficha de Personagens



Criar Ficha de Inimigo:



Gerenciar Grupo de Inimigo:



Gerenciar Grupo de Personagem:

