1. Descrição do Escopo do Sistema

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 conterão 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.

Embora não faça parte do escopo atual, há a previsão de futuras funcionalidades adicionais, como o compartilhamento de fichas via nuvem, integração com modo multiplayer, exportação de logs de batalhas e personalização avançada das regras. Esses recursos poderão ser implementados em versões futuras, de acordo com a evolução do projeto e o feedback dos usuários.

LEVANTANDO FUNCIONALIDADES:

O sistema proposto é um aplicativo desenvolvido em Flutter para Windows 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.
- Manter as fichas de personagens salvas em um banco de dados.
- Manter as fichas de monstros salvas em um banco de dados.
- Adicionar e manter cenários no banco de dados.
- Permitir a construção de batalhas ao selecionar grupos de fichas de personagens, 1 ou mais monstros e um cenário.
- Gerenciar as batalhas.

2. Descrição dos requisitos arquiteturais, objetivos e restrições de arquitetura:

- Única base de código em Flutter/Dart, com layouts adaptativos para web, tablet e celular (Android e IOS).
- Aplicação escalável horizontalmente
- Uso de pacotes Flutter/Dart confiáveis para criptografia, proteção de endpoints com TLS.

- Navegação guiada, suporte a temas claros/escuros, widgets customizados seguindo material design e cupertino.

Objetivos:

- Padrão do projeto - MVC, para facilitar extensões e correções.
- Priorizar testabilidade, suportando testes automatizados em criação de personagens, combates e rolagens de dados. Estrutura de testes unitários e de integração,
- Assegurar flexibilidade para expansões futuras, como novos modos de jogo ou integração com sistemas externos.

Restrições:

- Garantir compatibilidade com dispositivos móveis(Android e IOS) e navegadores modernos (Chrome, Firefox, Safari).
- As operações críticas do sistema devem ter tempo de resposta baixo.

3. Definição dos Padrões Arquiteturais Adotados:

O sistema adotará o padrão MVC (Model-View-Controller) para separar as responsabilidades em três camadas: o modelo (responsável pelos dados e pela lógica de negócio), a visão (responsável pela apresentação dos dados ao usuário) e o controlador (responsável por receber as requisições do usuário e atualizar o modelo e a visão). Também será utilizado o padrão Repository para fornecer uma interface abstrata para o acesso aos dados, facilitando a troca do mecanismo de persistência (banco de dados, armazenamento em nuvem).

O padrão Repository será utilizado para criar uma camada de abstração entre o aplicativo e o sistema de armazenamento de dados (banco de dados ou armazenamento em nuvem). Isso significa que o aplicativo não acessará o banco de dados diretamente, mas sim através de uma interface fornecida pelo Repository. Essa interface define métodos para realizar operações de leitura e escrita de dados, como criar, ler, atualizar e excluir.

A vantagem de usar o padrão Repository é que ele desacopla o aplicativo do sistema de armazenamento de dados. Isso facilita a troca do banco de dados, por exemplo, de um banco de dados relacional para um banco de dados NoSQL, sem a necessidade de alterar o código do aplicativo. Basta criar um novo Repository que implemente a mesma interface e que se comunique com o novo banco de dados.

4. Justificativa.

1. 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 banco de dados seja substituído ou reestruturado sem impactar diretamente outras partes do sistema.

---

2. Escalabilidade

O sistema foi concebido com foco em escalabilidade horizontal, permitindo que a aplicação cresça conforme a demanda por meio da replicação de instâncias em múltiplos dispositivos ou ambientes. 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.

---

3. 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.

---

4. Testabilidade

A estruturação do sistema segundo os princípios do MVC e o uso de injeção de dependência nos Repositories favorecem a criação de testes automatizados, tanto unitários quanto de integração. Cada componente pode ser testado isoladamente, garantindo que as regras de negócio, cálculos de combate e manipulação de dados estejam corretos mesmo antes da integração total.

---

5. Segurança

Embora o sistema não inclua integração com redes externas no escopo atual, a arquitetura já contempla o uso de pacotes Flutter/Dart confiáveis para criptografia e segurança de dados, com suporte a TLS para futuras conexões seguras com servidores externos. Isso prepara o sistema para a evolução planejada, como compartilhamento em nuvem e multiplayer.

---

6. 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.

---

7. 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. A arquitetura também considera o uso eficiente de recursos computacionais ao evitar chamadas de rede ou operações pesadas desnecessárias.

---

8. 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, como modos de jogo, integração com serviços externos e exportação de dados, sem exigir reestruturações profundas na base de código atual.