UNIVERSIDADE [Centro Universitário Católica de Santa Catarina]

Curso de Engenharia de Software

Sistema SaaS para Gerenciamento de Clãs em Jogos Online

Mateus Sales de Oliveira

Data de Entrega: [XX/XX/XXXX]

Orientador: [Nome do Professor]

Resumo

Este documento apresenta a proposta de desenvolvimento de um **SaaS (Software como Serviço) para gerenciamento de clãs em MMORPGs**, proporcionando uma plataforma **escalável, segura e automatizada** para auxiliar líderes e membros de clãs na **organização de eventos, gestão de membros, controle financeiro e distribuição de recompensas**.

A administração de clãs nos jogos online é, atualmente, um processo descentralizado e realizado manualmente, utilizando planilhas, mensagens instantâneas e ferramentas não especializadas. Isso pode gerar desorganização, dificultar a comunicação entre os membros e comprometer a eficiência da gestão dos recursos do clã. A solução proposta visa centralizar essas funções, garantindo um ambiente mais eficiente e estruturado para os líderes e participantes.

1 Introdução

1.1 Contexto

Os jogos **MMORPG (Massively Multiplayer Online Role-Playing Games)** envolvem a criação de **clãs**, que são grupos organizados de jogadores que cooperam para conquistar objetivos comuns, como batalhas, eventos e economia compartilhada. Atualmente, a gestão desses clãs é **manual e descentralizada**, dificultando a organização eficiente.

1.2 Justificativa

A falta de um **sistema estruturado para gerenciar membros, organizar eventos e distribuir recompensas** gera desorganização e possíveis conflitos. O projeto propõe uma **solução automatizada**, melhorando a administração do clã e aprimorando a experiência dos jogadores.

1.3 Objetivos

- Desenvolver um **SaaS** para facilitar a **gestão de clãs em MMORPGs**.
- Criar um **sistema onde líderes possam criar eventos** e permitir a confirmação de presença dos membros.
- Criar um **sistema onde os líderes possam criar um evento específico para distribuição de recompensas** em outro dia.
- Criar um **sistema de gestão financeira do clã**.
- Criar um **sistema de gerenciamento de builds** para organização estratégica de equipamentos.
- Desenvolver um **dashboard personalizável**.
- Integrar-se com **APIs de jogos e plataformas como Discord**.

2 Descrição do Projeto

2.1 Tema do Projeto

O projeto propõe o desenvolvimento de um **SaaS (Software como Serviço) para Gerenciamento de Clãs em Jogos Online (MMORPGs)**, permitindo que líderes e membros de clãs tenham acesso a ferramentas que facilitam a administração de eventos, controle de membros, gerenciamento financeiro e comunicação. A solução será uma **plataforma centralizada, acessível via web**, que permitirá aos jogadores **organizar atividades do clã de forma eficiente**, automatizando processos que atualmente são feitos de maneira manual e descentralizada.

A aplicação contará com um **painel administrativo completo**, no qual os líderes poderão **criar eventos, cadastrar builds para diferentes atividades do jogo, registrar transações financeiras e distribuir recompensas**. Além disso, haverá **integração com APIs externas** de jogos como Albion Online e plataformas de comunicação como Discord, possibilitando uma experiência dinâmica e interativa.

2.2 Problemas a Resolver

Atualmente, a gestão de clãs nos MMORPGs ocorre de forma descentralizada, utilizando planilhas, aplicativos de mensagens instantâneas e sistemas informais. Isso gera diversos desafios, incluindo:

- **Falta de ferramentas centralizadas**: A maioria dos clãs utiliza múltiplas plataformas para organização, tornando o gerenciamento confuso e ineficiente.
- **Dificuldade na organização de eventos**: Não há um sistema integrado para agendamento e confirmação de presença em eventos do clã.
- **Distribuição de recompensas manual e propensa a erros**: A alocação de loot ou ouro é feita sem um critério claro e pode causar conflitos entre membros.
- **Ausência de um sistema financeiro estruturado**: A administração de fundos do clã depende de registros informais, dificultando a transparência e controle de gastos.
- **Falta de um repositório para builds e estratégias**: Os jogadores pre- cisam registrar suas builds manualmente, o que dificulta a padronização.

- **Comunicação fragmentada**: A falta de integração com sistemas como Dis- cord obriga os membros a alternarem entre várias plataformas.
- **Dificuldade em obter dados do mercado do jogo**: A consulta de preços e economia dentro do jogo exige que os jogadores acessem fontes externas.

2.3 Limitações

Embora o sistema ofereça diversas funcionalidades para facilitar a gestão de clãs, algumas restrições foram estabelecidas para manter o escopo viável dentro do tempo e recursos disponíveis:

- **Foco inicial em Albion Online**: O SaaS será desenvolvido inicialmente para clãs de Albion Online, mas poderá ser expandido para outros MMORPGs futuramente.
- **Integrações limitadas na fase inicial**: A primeira versão incluirá integração com a API de mercado do Albion Online e Discord, mas outras APIs poderão ser adicionadas posteriormente.
- **Acesso limitado por níveis de permissão**: Nem todos os membros terão acesso irrestrito às funções do sistema, garantindo maior controle e segurança na administração do clã.
- **Otimização para dispositivos desktop e mobile**: O sistema será respon- sivo, mas **não incluirá um aplicativo mobile nativo na versão inicial**.

3 Especificação Técnica

3.1 Requisitos de Software

3.1.1 Requisitos Funcionais

Os requisitos funcionais definem as funcionalidades principais que o sistema deve fornecer para garantir uma gestão eficiente dos clãs em MMORPGs. A seguir, estão detalhados os requisitos essenciais:

- **RF01 - Cadastro e gerenciamento de membros:** O sistema deve permitir o cadastro, edição, remoção e visualização de membros do clã, incluindo informações como nome, cargo, nível de acesso e status de atividade.
- **RF02 - Criação e gerenciamento de eventos:** O sistema deve possibilitar que os líderes criem eventos com detalhes como data, horário, descrição, participantes e confirmação de presença.
- **RF03 - Gestão financeira do clã:** Deve ser possível registrar entradas e saídas financeiras, além de permitir a categorização de despesas e arrecadações do clã.
- **RF04 - Dashboard personalizável:** Cada usuário deve ter acesso a um painel com as informações mais relevantes para sua função no clã, podendo personalizar os widgets exibidos.
- **RF05 - Integração com APIs externas:** O sistema deve ser capaz de integrar-se com a API do mercado do Albion Online para consulta de preços e economia do jogo, além de se conectar ao Discord para envio de notificações automatizadas.
- **RF06 - Controle de permissões para diferentes cargos no clã:** Deve haver um sistema de hierarquia que permita atribuir permissões específicas para cada cargo, garantindo que somente os líderes e administradores tenham acesso a funções críticas.
- **RF07 - Histórico detalhado de transações financeiras do clã:** O sistema deve manter um registro de todas as movimentações financeiras, per- mitindo consultas e auditoria dos valores arrecadados e distribuídos.
- **RF08 - Relatórios detalhados sobre eventos e economia do clã:** Deve haver a geração automática de relatórios que exibam estatísticas sobre eventos realizados, participação dos membros e análise financeira do clã.

- **RF09 - Criação e gerenciamento de builds personalizadas:** O sistema deve permitir que líderes e membros autorizados cadastrem builds (conjuntos de equipamentos e habilidades) para diferentes tipos de eventos.
- **RF10 - Agendamento de eventos recorrentes:** Deve ser possível programar eventos que ocorram regularmente, como reuniões semanais ou treinos estratégicos.
- **RF11 - Sistema de notificações e alertas:** O sistema deve enviar alertas automáticos para os membros sobre eventos futuros, mudanças de planos ou movimentações financeiras do clã.
- **RF12 - Exportação de dados:** Os administradores do clã devem ter a opção de exportar listas de membros, histórico de eventos e registros financeiros em formatos como CSV ou PDF.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais definem as características técnicas e de qualidade que o sistema deve atender para garantir um funcionamento eficiente, seguro e confiável.

- **RNF01 - Alta disponibilidade e escalabilidade:** O sistema deve suportar múltiplos usuários simultaneamente e estar disponível pelo menos 99,5% do tempo.
- **RNF02 - Autenticação segura e criptografia de dados:** Todas as autenticações devem ser realizadas via OAuth 2.0 ou JWT, e os dados sensíveis devem ser criptografados usando AES-256.
- **RNF03 - Compatibilidade com dispositivos móveis e desktop:** O sistema deve ser responsivo e acessível em diferentes resoluções, garantindo boa experiência tanto em computadores quanto em smartphones e tablets.
- **RNF04 - Interface responsiva e intuitiva:** O design deve seguir princípios de UI/UX modernos para proporcionar uma navegação fluida e intuitiva para os usuários.
- **RNF05 - Tempo de resposta das operações críticas inferior a 300ms:** Todas as operações essenciais, como login, carregamento do painel e execução de comandos administrativos, devem ter um tempo de resposta inferior a 300ms em condições normais de operação.
- **RNF06 - Logs detalhados para auditoria de operações do sistema:** Todas as ações realizadas pelos usuários no sistema devem ser registradas em

logs de auditoria, incluindo informações sobre acesso, alterações de dados e movimentações financeiras.

- **RNF07 - Suporte a múltiplos idiomas:** O sistema deve permitir a troca de idioma na interface para atender a usuários de diferentes nacionalidades.
- **RNF08 - Backup automático dos dados:** O sistema deve realizar back- ups periódicos para evitar perda de informações em caso de falha ou ataque cibernético.
- **RNF09 - Integração com serviços de monitoramento:** A infraestrutura do sistema deve contar com ferramentas de monitoramento como Prometheus e Grafana para análise de desempenho e detecção de falhas.
- **RNF10 - Compatibilidade com CI/CD:** O código do sistema deve seguir padrões que permitam a automação de deploys e testes contínuos via pipelines como GitHub Actions ou Jenkins.

3.2. Considerações de Design

3.2.1 Discussão sobre as Escolhas de Design

As decisões de design foram tomadas com base na necessidade de um sistema escalável, modular e de fácil manutenção. Para isso, foram analisadas algumas alternativas:

- **Arquitetura Monolítica:** Embora seja mais simples de implementar, dificultaria a escalabilidade e a separação de responsabilidades.
- **Arquitetura de Microserviços:** Escolhida por permitir maior escalabilidade, divisão modular e manutenção mais eficiente.
- **Banco de Dados Relacional vs. Não Relacional:** PostgreSQL foi escolhido por oferecer suporte a consultas complexas e garantir integridade dos dados.
- **Autenticação Centralizada:** Uso de Firebase Auth/Autho para simplificar a implementação e garantir segurança.

3.2.2 Visão Inicial da Arquitetura

A arquitetura do sistema será composta pelos seguintes componentes principais:

- **Frontend (Vue.js):** Interface do usuário interativa e dinâmica.
- **Backend (NestJS):** API responsável por processar requisições e gerenciar lógica de negócio.
- **Banco de Dados (PostgreSQL + Redis):** Armazena dados estruturados e gerencia cache para otimização de desempenho.
- **Serviço de Autenticação (Firebase Auth/Autho):** Gerencia credenciais e permissões de usuários.
- **Serviço de Notificações (Discord API):** Integração com Discord para envio de alertas automáticos.
- **Monitoramento (Prometheus + Grafana):** Análise e visualização de métricas do sistema.

Esses componentes se comunicam via **APIs RESTful**, garantindo flexibilidade na integração com novos serviços.

3.2.3 Padrões de Arquitetura

Os padrões arquiteturais utilizados incluem:

- **Microserviços:** Para facilitar a escalabilidade e independência dos módulos.
- **MVC (Model-View-Controller):** Aplicado no frontend para organização do código.
- **Event-Driven Architecture:** Comunicação assíncrona entre serviços, garantindo melhor desempenho.
- **Repository Pattern:** Implementado no backend para desacoplar regras de negócio do banco de dados.

3.2.4 Modelos C4

A arquitetura será documentada em **Modelagem C4**, abrangendo:

- **Contexto:** Visão geral do sistema e suas interações externas.
- **Contêineres:** Representação dos principais serviços do sistema.
- **Componentes:** Detalhamento da comunicação entre microsserviços.
- **Código:** Estrutura interna das principais classes e módulos.

3.3 Stack Tecnológica

3.3.1 Linguagens de Programação

A escolha das linguagens de programação foi baseada na **eficiência, escalabilidade e suporte da comunidade**. As tecnologias escolhidas são:

- **TypeScript:** Utilizado no **backend (NestJS)** e **frontend (Vue.js)**, trazendo segurança, organização e melhor manutenção do código devido à tipagem estática.
- **JavaScript:** Base para **TypeScript**, sendo amplamente suportado e eficiente para aplicações web escaláveis.
- **SQL (PostgreSQL):** Utilizado no **banco de dados relacional**, garantindo consistência e integridade dos dados.

3.3.2 Frameworks e Bibliotecas

A seleção de frameworks e bibliotecas foi feita para garantir **desempenho, escalabilidade e facilidade de desenvolvimento**:

Frontend

- **Vue.js:** Framework progressivo para construção de interfaces web modernas e dinâmicas.
- **Tailwind CSS:** Framework CSS baseado em classes utilitárias, garantindo agilidade e responsividade na estilização.

Backend

- **NestJS:** Framework para Node.js que utiliza **arquitetura modular**, seguindo princípios SOLID e garantindo escalabilidade.
- **Express.js:** Base do NestJS, utilizado para manipulação de rotas e API REST.
- **Axios:** Biblioteca para requisições HTTP, utilizada para comunicação entre frontend e backend.

Banco de Dados e Cache

- **PostgreSQL:** Banco de dados relacional escolhido por sua confiabilidade, escalabilidade e suporte a consultas complexas.
- **Redis:** Utilizado para **armazenamento em cache**, melhorando o desempenho da aplicação.

Autenticação e Segurança

- **Firebase Auth:** Serviço gerenciado para autenticação de usuários, suportando login via Google, Discord e email/senha.
- **Autho:** Alternativa para autenticação segura, escalável e com suporte a múltiplos métodos de login.

3.3.3 Ferramentas de Desenvolvimento e Gestão de Projeto

- **Versionamento de Código:**
    - **Git + GitHub:** Controle de versão e colaboração entre desenvolvedores.
    - **GitHub Actions:** Automatização de testes e integração contínua (CI/CD).
- **Gerenciamento de Infraestrutura:**
    - **Docker:** Criação e gestão de containers, garantindo compatibilidade e portabilidade da aplicação.
    - **Kubernetes:** Orquestração de containers para escalabilidade e alta disponibilidade.
- **Monitoramento e Observabilidade:**
    - **Prometheus:** Coleta de métricas e monitoramento da infraestrutura.
    - **Grafana:** Visualização de métricas e logs para análise de desempenho.
- **Gerenciamento de Projetos:**
    - **Azure DevOps / GitHub Projects:** Organização de tarefas utilizando metodologia **Kanban**.
    - **Notion / Wiki do GitHub:** Documentação colaborativa do projeto.

3.3.4 Integrações com APIs Externas

Para enriquecer a experiência dos usuários, o SaaS contará com integrações estratégicas:

- **Albion Online API:** Permite acesso a dados do jogo, como preços de mercado e estatísticas de jogadores.
- **Discord API:** Notificações automáticas para membros do clã sobre eventos e atualizações importantes.

4 Próximos Passos

- Refinamento da modelagem e arquitetura do sistema.
- Desenvolvimento incremental e testes contínuos.
- Implementação de autenticação e autorização.
- Integração de APIs de jogos e sistemas externos.
- Validação do MVP (Produto Mínimo Viável) com usuários reais.

5 Referências

- **FOWLER, Martin.** *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- Documentação oficial do **PostgreSQL**: https://www.postgresql.org/docs/
- Documentação oficial do **Node.js**: https://nodejs.org/en/docs/
- Documentação oficial do **Vue.js**: https://vuejs.org/guide/introduction.html
- Documentação da API de **Albion Online**: https://www.albion-online-data.com/
- Documentação da API do **Discord**: https://discord.com/developers/docs/intro
- Biblioteca oficial do **Express.js**: https://expressjs.com/
- Biblioteca oficial do **Axios**: https://axios-http.com/
- Biblioteca oficial do **Tailwind CSS**: https://tailwindcss.com/docs

6 Apêndices (Opcionais)

Espaço para inserção de anexos e informações complementares.

7 Avaliações de Professores

Considerações Professor/a:

Considerações Professor/a:

Considerações Professor/a: