**Documentação do Projeto: ChatBot Fúria**

**Visão Geral**

O projeto é uma aplicação web de chat que combina:

1. Um chatbot especializado em informações sobre a equipe de eSports Fúria
2. Um sistema de mensagens entre usuários
3. Uma plataforma de comunicação em tempo real via WebSocket

**Tecnologias Utilizadas**

***Backend***

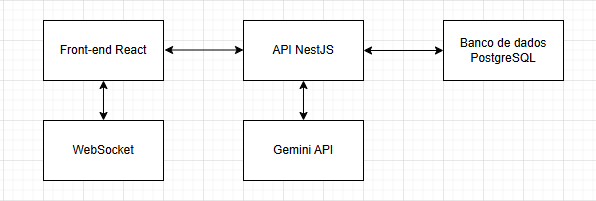
* **NestJS**: Framework para construção da API
* **Prisma**: ORM para interação com o banco de dados
* **PostgreSQL**: Banco de dados relacional
* **WebSocket**: Para comunicação em tempo real
* **Google Gemini API**: Integração com IA generativa

***Frontend***

* **React**: Biblioteca para construção da interface
* **TypeScript**: Tipagem estática
* **Socket.io-client**: Cliente WebSocket
* **Axios**: Cliente HTTP

**Arquitetura do Sistema**

***Diagrama de Componentes***

******

**Funcionalidades Principais**

**1. Sistema de Autenticação**

* Registro de novos usuários
* Login com email e senha
* Armazenamento de sessão

**2. ChatBot Fúria**

* Respostas automáticas sobre:
  + Próximos jogos da Fúria
  + Elenco atual
  + Resultados recentes
* Personalidade "furiosa" com respostas em CAPSLOCK e emojis
* Integração com HLTV.org para dados atualizados

**3. Sistema de Mensagens**

* Chat entre usuários
* Status online/offline
* Histórico de mensagens
* Notificações em tempo real

**4. Gerenciamento de Amigos**

* Adição de amigos por email
* Listagem de conversas
* Status de conexão

**Modelagem de Dados**

**Diagrama ER**

*User (1) ↔ (n) UserConnection (n) ↔ (1) User*

*User (1) ↔ (n) Message (n) ↔ (1) User*

**Modelos Prisma**

* **User**: Armazena informações dos usuários
* **UserConnection**: Representa relações de amizade
* **Message**: Armazena mensagens trocadas

**Fluxos Principais**

**1. Autenticação**

1. Usuário se registra ou faz login
2. Todas as requisições subsequentes incluem o token

**2. ChatBot**

1. Usuário envia pergunta
2. Sistema consulta:
   * API HLTV para dados atualizados
   * Google Gemini para gerar resposta no estilo "furioso"
3. Resposta é exibida no chat

**3. Mensagens entre Usuários**

1. Usuário seleciona um amigo
2. Mensagens são carregadas do banco de dados
3. Novas mensagens são enviadas via WebSocket
4. Mensagens são entregues em tempo real

**Configuração do Ambiente**

**Requisitos**

* Node.js (v18+)
* PostgreSQL
* Conta no Google AI Studio (para chave da API Gemini)

**Variáveis de Ambiente**

DATABASE\_URL=postgresql://user:password@localhost:5432/dbname

GEMINI\_API\_KEY=sua-chave-aqui

**Endpoints da API**

**Autenticação**

* POST /user - Registrar novo usuário
* POST /user/login - Fazer login

**Usuários**

* GET /user/search - Buscar usuários

**Amigos**

* POST /friends/:userId/start-conversation - Iniciar conversa
* GET /friends/:userId/conversations - Listar conversas

**Mensagens**

* GET /messages/:userId/:friendId - Obter histórico de mensagens

**ChatBot**

* POST /furia/quest - Enviar pergunta ao bot

**WebSocket Events**

*Recebidos*

* sendMessage - Enviar nova mensagem

*Emitidos*

* newMessage - Nova mensagem recebida
* friendStatusChanged - Atualização de status de amigo

**Conclusão**

Este projeto combina tecnologias modernas para criar uma experiência de chat envolvente para fãs da Fúria, com um chatbot único e funcionalidades de mensagens em tempo real. A arquitetura foi projetada para ser escalável e manterável, com clara separação de responsabilidades entre os componentes.