UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO

Sistemas Distribuídos
Rafael Oliveira Vasconcelos

Escopo inicial, arquitetura e tecnologias

João Pedro Cardoso Arruda
Nicolas Matheus Ferreira de Jesus
Vitor Leonardo Sena de Lima

SÃO CRISTÓVÃO
05/12/2025

Tema, Gênero, Regras e Escopo do jogo

Tema e Gênero: Futebol online top-down com física simples de empurrão. Gênero: Esporte Multiplayer em Tempo Real.
Premissas de rede: Baixa latência, servidor autoritativo e sincronização constante de estado via WebSockets.

Regras principais (confirmadas em código):

- **Objetivo:** Marcar mais gols que o adversário dentro do tempo de partida (`game/gameLoop.js` gols + cooldown; `game/match.js` fim/reinício).
- **Times:** Vermelho e Azul (`game/roomManager.js`, `game/match.js`).
- **Início:** Só começa com pelo menos 1 jogador em cada time; pausa/espera se um time ficar vazio (`game/match.js` checkRestartConditions).
- **Capacidade:** Até 6 jogadores por sala (`game/constants.js` MAX_PLAYERS_PER_ROOM; `game/roomManager.js` alocação e sanitização de IDs).
- **Duração:** 60 segundos por partida (`game/constants.js` MATCH_DURATION; `game/match.js` updateTimer).
- **Campo e bola:** Campo 800x600; gols 50x200; bola raio 10 (`game/roomManager.js`, `game/constants.js`); cooldown de gol e reset de bola (`game/gameLoop.js`, `game/ball.js`).
- **Movimentação e colisões:** Servidor calcula colisões jogador-bola, paredes e cantos (`game/gameLoop.js` + `game/ball.js`); cliente só envia inputs e renderiza

(`public/game.js`).

Escopo Inicial (Arquitetura Proposta e Tecnologias)

Jogo web 2D em tempo real, modelo **cliente-servidor autoritativo**: o servidor mantém o estado oficial (física, placar, timer); os clientes enviam comandos e renderizam o snapshot recebido.

Arquitetura (atual, conforme código)

- **Cliente (browser):**
  - HTML5 + Canvas para render (`public/index.html`).
  - JavaScript (DOM/Canvas) para entrada, HUD e mobile joystick (`public/game.js` ; `public/style.css`).
  - Socket.IO Client para comunicação em tempo real (`public/index.html` + `public/game.js`).
- **Servidor de jogo (Node.js):**
  - Node.js + Express para servir estáticos e endpoint Socket.IO (`game-server.js`).
  - Socket.IO Server para canal bidirecional (`game-server.js`).
  - Loop de jogo a 60 FPS (`setInterval` 1000/60 em `game-server.js` ; lógica em `game/gameLoop.js`).
  - Timer de partida a cada 1s (`setInterval` em `game-server.js` ; `updateTimer` em `game/match.js`).
  - Autoridade total sobre estado: posições, bola, placar, tempo (`game/gameLoop.js` , `game/match.js` , `game/ball.js`).
- **Salas:**
  - Criação/alocação dinâmica com ID sanitizado (`game/roomManager.js` sanitizeRoomId, allocateRoom).
  - Máx. 6 jogadores/sala; balanceamento automático (`game/constants.js` ; `game/match.js` balanceTeams).
  - Limpeza de sala quando vazia (`game/roomManager.js` cleanupRoomIfEmpty).
- **Infraestrutura / Deploy:**
  - Execução local simples (Node 18+).
  - **Dockerfile** para build da aplicação Node (root `dockerfile`).
  - **Docker Compose** opcional com Nginx como reverse proxy (porta 80 → app:3000) (`docker-compose.yml` , `nginx/Dockerfile`).
  - Hospedagem típica em VM ou containerizada (ex.: AWS EC2).

- PM2 e supervisores semelhantes não fazem parte do stack principal atual.

Tecnologias Principais

Componente	Tecnologia	Função Principal
Servidor de jogo	Node.js + Express	Lógica central, serving estáticos
Tempo real	Socket.IO (WebSockets)	Comunicação bidirecional de baixa latência
Cliente (render)	HTML5 Canvas + JS	Render do campo, jogadores, bola e HUD
Infra (proxy opc.)	Nginx	Reverse proxy para a aplicação Node
Containers	Docker / Docker Compose	Empacotamento e orquestração local/produção
Cloud (ex.)	AWS EC2	Hospedagem em VM ou containerizada

Pontos de Foco (próxima fase)

- **Sincronização de estado:** Loop de jogo (60 FPS) no servidor envia o snapshot oficial; cliente é controlador/visualização.
- **Autoridade do servidor:** Colisões, gols, cronômetro e placar calculados no backend para mitigar cheating.
- **Salas/sessões:** Até 6 jogadores; partida inicia com 1 por time; pausa se um time esvaziar; limpeza de sala vazia.
- **Latência e UX:** Pings periódicos exibem latência; suporte a teclado e controles móveis (joystick) (`public/game.js` , `public/style.css`).
- **Fluxos de partida:** Cooldown de gol, reset de bola e jogadores, término por tempo (60s), botão “Jogar Novamente” para reinício coordenado (`game/gameLoop.js` , `game/match.js` , `game/ball.js`).

Ideias Futuras

- Exibir ping no HUD (pings já emitidos do servidor — `game/socketHandlers.js` ; mostrar de forma visível no cliente).

- Associar um identificador/nome ao usuário (ex.: derivar de IP ou outro identificador) para exibir próximo ao boneco/placar.
- Criar ranking de jogadores (gols, vitórias, partidas); persistir e ordenar.
- Adotar cache para dados de ranking/estado (ex.: Redis) para reduzir latência e custo de recomputação.
- Distribuir a aplicação em múltiplas instâncias (escalar horizontalmente) — requererá sticky sessions ou compartilhamento de estado (ex.: Redis/adaptador Socket.IO) para manter coesão entre instâncias.