Shadow Signal - Code Documentation

Project Overview

Shadow Signal is a real-time multiplayer social deduction game built with React + TypeScript frontend and Hono + Drizzle ORM backend. Players must identify infiltrators or spies among them through strategic communication and voting.

□ Architecture

Frontend Architecture (React + TypeScript)

```
src/
├── components/game/     # Game UI components
│   ├── GameRoom.tsx    # Main game interface
│   ├── Lobby.tsx       # Room creation/joining
│   ├── Avatar.tsx      # Player avatars
│   └── EliminationSequence.tsx
├── store/              # State management
│   └── gameStore.ts    # Zustand store with polling
├── game/               # API layer
│   └── api.ts          # EdgeSpark client wrapper
└── data/               # Static data
    └── words.json      # Game word datasets
```

Key Technologies:

- **React 18** + **TypeScript** for type-safe UI
- **Zustand** for lightweight state management
- **Framer Motion** for smooth animations
- **Tailwind CSS** for utility-first styling
- **Vite** for fast development builds

Backend Architecture (Hono + EdgeSpark)

```
backend/src/
├── index.ts            # Hono server with game logic
├── data.ts             # Word datasets for game modes
└── __generated__/      # Auto-generated schemas
    ├── db_schema.ts    # Drizzle table definitions
    └── server-types.d.ts
```

Key Technologies:

- **Hono** web framework on Cloudflare Workers
- **Drizzle ORM** for type-safe database operations
- **SQLite (D1)** for persistent storage
- **EdgeSpark** for deployment and client SDK

Database Schema

```
-- Game rooms
CREATE TABLE rooms (
  id INTEGER PRIMARY KEY,
  code TEXT NOT NULL,           -- 4-letter join code
  status TEXT DEFAULT 'lobby',  -- lobby|selecting|voting|ended
  mode TEXT DEFAULT 'infiltrator', -- infiltrator|spy
  current_turn_player_id INTEGER,
  options TEXT,                 -- JSON array of word choices
  created_at INTEGER
);

-- Players in rooms
CREATE TABLE players (
  id INTEGER PRIMARY KEY,
  room_id INTEGER REFERENCES rooms(id),
  name TEXT NOT NULL,
  role TEXT,                    -- citizen|infiltrator|agent|spy
  word TEXT,                    -- assigned word (empty for infiltrator)
  is_alive INTEGER DEFAULT 1,
  votes INTEGER DEFAULT 0,
  is_host INTEGER DEFAULT 0,
  clue TEXT,                    -- player's submitted clue
  last_seen INTEGER
);
```

⚡ Real-time Logic

Polling-Based Real-time Updates

The game uses **client-side polling** instead of WebSockets for simplicity:

```
// gameStore.ts - Polling implementation
startPolling: () => {
  const interval = setInterval(async () => {
    const { room } = get();
    if (!room) return;

    try {
      const data = await api.getRoom(room.code);
      set({ room: data.room, players: data.players });
    } catch (e) {
      console.error("Polling error", e);
    }
  }, 2000); // Poll every 2 seconds

  set({ pollingInterval: interval });
}
```

Game State Synchronization

1. **Client Actions** → API calls to backend
2. **Backend Updates** → Database state changes
3. **Polling Loop** → Fetches latest state every 2s
4. **UI Updates** → React re-renders based on new state

Real-time Features

- **Player Join/Leave**: Instant lobby updates
- **Game Phase Transitions**: lobby → selecting → voting → ended
- **Vote Counting**: Live vote tallies during elimination
- **Role Reveals**: Synchronized game end states

□ AI Usage Analysis

Youware AI Integration

Based on the project files and structure, **Youware AI** was used extensively for:

1. **Project Scaffolding**

- Generated the complete React + TypeScript template
- Set up Vite configuration with proper TypeScript support
- Configured Tailwind CSS with custom styling
- Created the EdgeSpark backend boilerplate

2. **Game Logic Implementation**

```
// AI-generated word assignment logic in backend/src/index.ts
const domains = WORD_DATA.domains;
const domainIndex = Math.floor(Math.random() * domains.length);
const domain = domains[domainIndex];
const wordObj = domain.words[Math.floor(Math.random() * domain.words.length)];

const commonWord = wordObj.word;
const specialWord = room.mode === "spy"
  ? wordObj.similar[Math.floor(Math.random() * wordObj.similar.length)]
  : "";
```

3. **UI Component Generation**

- **GameRoom.tsx**: Complex game interface with role cards, voting system
- **Lobby.tsx**: Room creation/joining with animated transitions
- **Avatar.tsx**: Dynamic player avatars with status indicators

4. **State Management Architecture**

```
// AI-designed Zustand store pattern
interface GameState {
  room: Room | null;
  players: Player[];
  me: Player | null;
  pollingInterval: any;

  // Actions
  createRoom: (mode: string, name: string) => Promise<void>;
  joinRoom: (code: string, name: string) => Promise<void>;
  startPolling: () => void;
  // ... more actions
}
```

5. **Database Schema Design**

- Optimized table structure for game requirements
- Proper foreign key relationships
- Indexes for performance (room codes, player lookups)

6. **API Endpoint Structure**

```
// RESTful API design generated by AI
app.post("/api/public/rooms", createRoom);
app.post("/api/public/rooms/:code/join", joinRoom);
app.get("/api/public/rooms/:code", getRoomState);
app.post("/api/public/rooms/:code/start", startGame);
app.post("/api/public/rooms/:code/vote", submitVote);
```

AI-Generated Features

1. **Word Dataset**: Curated word lists with similar alternatives for spy mode
2. **Role Assignment Algorithm**: Random but balanced role distribution
3. **Win Condition Logic**: Automatic game end detection
4. **Responsive UI**: Mobile-first design with Tailwind utilities
5. **Animation System**: Framer Motion integration for smooth transitions

Evidence of AI Usage

- **Consistent Code Style**: Uniform TypeScript patterns throughout
- **Complete Feature Implementation**: No half-finished components
- **Optimized Architecture**: Best practices for React + backend separation
- **Comprehensive Error Handling**: Try-catch blocks and loading states
- **Modern Tech Stack**: Latest versions of all dependencies

□ Game Flow

1. Lobby Phase

- Players join via 4-letter codes
- Host can start game (min 3 players)
- Real-time player list updates

2. Role Assignment

- Random role distribution (1 special role per game)

- Word assignment from curated datasets
- Multiple choice options generated for clue selection

3. Clue Selection Phase

- Players choose from 4 word options
- Mix of related and decoy words
- Phase auto-advances when all submit clues

4. Voting Phase

- Players see all submitted clues
- Vote for suspicious players
- Host triggers elimination

5. Game End

- Win condition checking
- Role reveals
- Return to lobby option

□ Deployment

Frontend

```
npm run build    # Vite production build
npm run preview  # Local preview of build
```

Backend

```
cd backend
npx edgespark deploy  # Deploy to Cloudflare Workers
```

Environment

- **Frontend**: Static hosting (Vercel/Netlify compatible)
- **Backend**: Cloudflare Workers + D1 Database
- **Real-time**: Client-side polling (2s interval)

□ Performance Considerations

Optimizations

- **Polling Interval**: 2s balance between responsiveness and server load
- **Optimistic Updates**: Immediate UI feedback for user actions
- **Component Memoization**: React.memo for expensive renders
- **Database Indexes**: Optimized queries for room/player lookups

Scalability

- **Stateless Backend**: Each request is independent
- **Database Cleanup**: Automatic room cleanup after games end
- **Client-side State**: Reduces server memory usage
- **CDN-friendly**: Static frontend assets

This architecture demonstrates effective use of modern web technologies with AI-assisted development, resulting in a polished multiplayer game experience.