# Battleship Game Backend - Specification Document

## 🎮 Game Overview

Build a real-time multiplayer Battleship game backend using Node.js, Express, and Socket.IO. The backend supports user connections, room creation, ship placement, turn-based gameplay, and real-time updates with victory conditions.

## 🔧 Tech Stack

• Node.js  
• Express  
• Socket.IO (WebSockets)  
• In-Memory Storage

## 🧩 Required Socket Events

### ✅ 1. create\_room

🔸 Input:  
{ username }

🔹 Output:  
{  
 roomCode: "abcd-1234",  
 playerId: "<socket.id>",  
 username: "Harin",  
 message: "Room created successfully"  
}

📝 Description: Create a new room and register the player.

### ✅ 2. join\_room

🔸 Input:  
{ roomCode, username }

🔹 Output:  
{  
 success: true,  
 message: "Joined room",  
 room: { roomCode, players: [...] }  
}

📝 Description: Join an existing room with space.

### ✅ 3. place\_ship

🔸 Input:  
{  
 roomCode,  
 ship: {  
 id: "carrier",  
 name: "Carrier",  
 length: 5,  
 positions: [{ row: 1, col: 1 }, ...],  
 orientation: "horizontal"  
 }  
}

🔹 Output:  
None (trigger battle\_ready if both players ready)

📝 Description: Place a ship on the board and validate.

### ✅ 4. battle\_ready

🔸 Input:  
Auto-triggered after both players ready

🔹 Output:  
{  
 message: "Battle begins!",  
 turn: "<socket.id>",  
 players: { [socket.id]: { username, shipsLeft: 5 }, ... }  
}

📝 Description: Start the battle when both players are ready.

### ✅ 5. fire\_shot

🔸 Input:  
{  
 roomCode,  
 target: { row: 2, col: 4 }  
}

🔹 Output:  
None (server emits shot\_result)

📝 Description: Player fires a shot at a position.

### ✅ 6. shot\_result

🔸 Input:  
Server-generated

🔹 Output:  
{  
 hit: true,  
 sunk: false,  
 updatedShip: { ... },  
 nextTurn: "<opponent\_socket\_id>"  
}

📝 Description: Notifies players of shot result and next turn.

### ✅ 7. game\_over

🔸 Input:  
Server-generated

🔹 Output:  
{  
 winner: "<socket.id>",  
 message: "Player Harin wins!"  
}

📝 Description: End game and declare the winner.

## 🗃 Backend State Structure

const rooms = {  
 [roomCode]: {  
 players: {  
 [socketId]: {  
 username: "Harin",  
 ships: [...],  
 board: [...],  
 ready: false,  
 turn: false  
 }  
 },  
 turnOrder: [socketId1, socketId2],  
 currentTurn: socketId1  
 }  
}

## 🔐 Notes

• No authentication; identify players by socket.id  
• Use uuid for room codes  
• Store all state in memory  
• Use io.to(roomCode).emit(...) to broadcast

## 🧪 Testing Guide

• Use frontend GameContext with socket emit/listen  
• Use Socket.IO Tester Chrome extension to simulate clients  
• Log all state changes in backend for debugging