1. Architecture

Use client-server architecture.
Reason: easiest to secure, no desync, best for turn-based games.
No data exchange between players -> P2P is not necessary

2. Message type

Use JSON-based protocol over TCP, divided into 5 categories:

- AUTH: registration, login, logout
- LOBBY: create/join/leave room, list room, list players
- GAME_ACTION: call, fold, switch, all-in
- GAME_STATE: server sends to client (cards revealed, new turn, final result)
- HISTORY: for browsing player history
- SYSTEM: error message, server notices

Every message ends with '\n' character and must follow the general format:

```
{
        "type": "STRING",
         ... other fields …
}
```

a. Auth message

- Client -> server

Message	Description	Format

register		{
		  "type": "AUTH_REGISTER",
		  "username": "alice",
		  "password": "1234"
		}

login		{
		  "type": "AUTH_LOGIN",
		  "username": "alice",
		  "password": "1234"
		}

logout		{
		  "type": "AUTH_LOGOUT"
		}

- Server -> client

Message	Description	Format

register result		{
		  "type":
		"AUTH_REGISTER_RESULT",
		  "success": true,
		  "reason": null
		}

login result		{
		  "type":
		"AUTH_LOGIN_RESULT",
		  "success": true,
		  "reason": null
		}

logout result		{
		  "type":
		"AUTH_LOGOUT_RESULT",
		  "success": true
		}

b. Lobby message

- Client -> server

Message	**Description**	**Format**

list all rooms		{
		  "type": "LOBBY_LIST"
		}

create room		{
		  "type": "LOBBY_CREATE",
		  "room_name": "Room A",
		  "private": false,
		  "password": ""
		}

join room		{
		  "type": "LOBBY_JOIN",
		  "room_id": 12,
		  "password": ""
		}

leave room		{
		  "type": "LOBBY_LEAVE"
		}

start game	The host starts the game	{
		  "type":
		"LOBBY_START_GAME"
		}

- Server -> client

Message	**Description**	**Format**
list all rooms		{

result		"type":
		"LOBBY_LIST_RESULT",
		"rooms": [
		{
		"room_id": 12,
		"name": "Poker Room",
		"players": 4,
		"private": false
		}
		]
		}

create room result		{
		"type":
		"LOBBY_CREATE_RESULT",
		"success": true,
		"room_id": 12,
		"reason": null
		}

join room result		{
		"type":
		"LOBBY_JOIN_RESULT",
		"success": true,
		"reason": null
		}

leave room result		{
		"type":
		"LOBBY_LEAVE_RESULT",
		"success": true
		}

c. **Game flow**

- Client -> server

Message	**Description**	**Format**

call		{
		"type": "GAME_CALL"
		}

fold		{
		"type": "GAME_FOLD"
		}

switch		{
		"type": "GAME_SWITCH",
		"replace_index": 0, // 0 or 1
		"option_index": 2 // 0, 1, or 2
		}

all in		{ "type": "GAME_ALL_IN" }

d. Game state

- Server -> client

Message	Description	Format
start game		{ "type": "GAME_START", "players": ["alice","bob","charlie"] }
private cards dealt		{ "type": "GAME_PRIVATE_CARDS", "cards": ["KH", "9D"] }
card switch offer		{ "type": "GAME_SWITCH_OFFER", "options": ["4D", "2S", "AH"] }
card switch result		{ "type":"GAME_SWITCH_RESULT", "replaced_index":0, "new_card":"AH" }
update risk level	Announces the current round risk Progression: - 1 (before private cards) - 2 (pre-flop) - 3 (flop) - 4 (turn) - 5 (river) - 8 (all-in)	{ "type": "GAME_RISK_LEVEL_UPDATE", "risk": 1 }
shared card update	Sent when cards are revealed each round	{ "type": "GAME_SHARED_UPDATE", "cards": ["3H","7C","QD"]

		}

action broadcast	Sent to everyone when a player acts	{
		  "type": "GAME_ACTION_BROADCAST",
		  "player": "alice",
		  "action": "CALL"
		}

showdown	Sent when all players have acted for the final stage, or an all-in occurs	{
		  "type": "GAME_SHOWDOWN",
		  "results": {
		    "winner": "bob",
		    "losers": [
		      { "player": "alice", "risk": 5 },
		      { "player": "charlie", "risk": 5 }
		    ]
		  },
		  "shared_cards": ["3H","7C","QD","TS","5H"]
		}

risk resolve	Roulette outcome after fold or showdown loss.	{
		  "type": "GAME_RISK_RESOLVE",
		  "player": "alice",
		  "risk": 3,
		  "survived": false,
		  "reason": "bullet_hit" // bullet_hit | safe
		}

spectator player	player is eliminated and can only spectate game	{
		  "type": "GAME_PLAYER_SPECTATOR",
		  "player": "alice"
		}

god save	Rare ‘God save’ after all in lost	{
		  "type": "GAME_PLAYER_REVIVED",
		  "player": "charlie",
		  "reason": "god_save"
		}

update state		{

		"type": "GAME_STATE_UPDATE", "players": [{"name":"alice", "status":"alive"}, {"name":"bob", "status":"alive"}, {"name":"charlie", "status":"spectator"}] }
end game		{ "type": "GAME_END", "winner": "bob" }

e. History messages

Message	**Description**	**Format**
get history (client->server)		{ "type":"HISTORY_GET", "limit":20 }
return history result (server->client)		{ "type": "HISTORY_RESULT", "games": [{ "timestamp": 1731440100, "room": "Room A", "winner": "bob", "players": ["alice","bob","charlie"], "rounds": 5 }] }

f. System message

Message	**Description**	**Format**
ping		{"type":"SYSTEM_PING"}
pong		{"type":"SYSTEM_PONG"}
system error		{ "type": "SYSTEM_ERROR", "reason": "spectators cannot act" }

		}