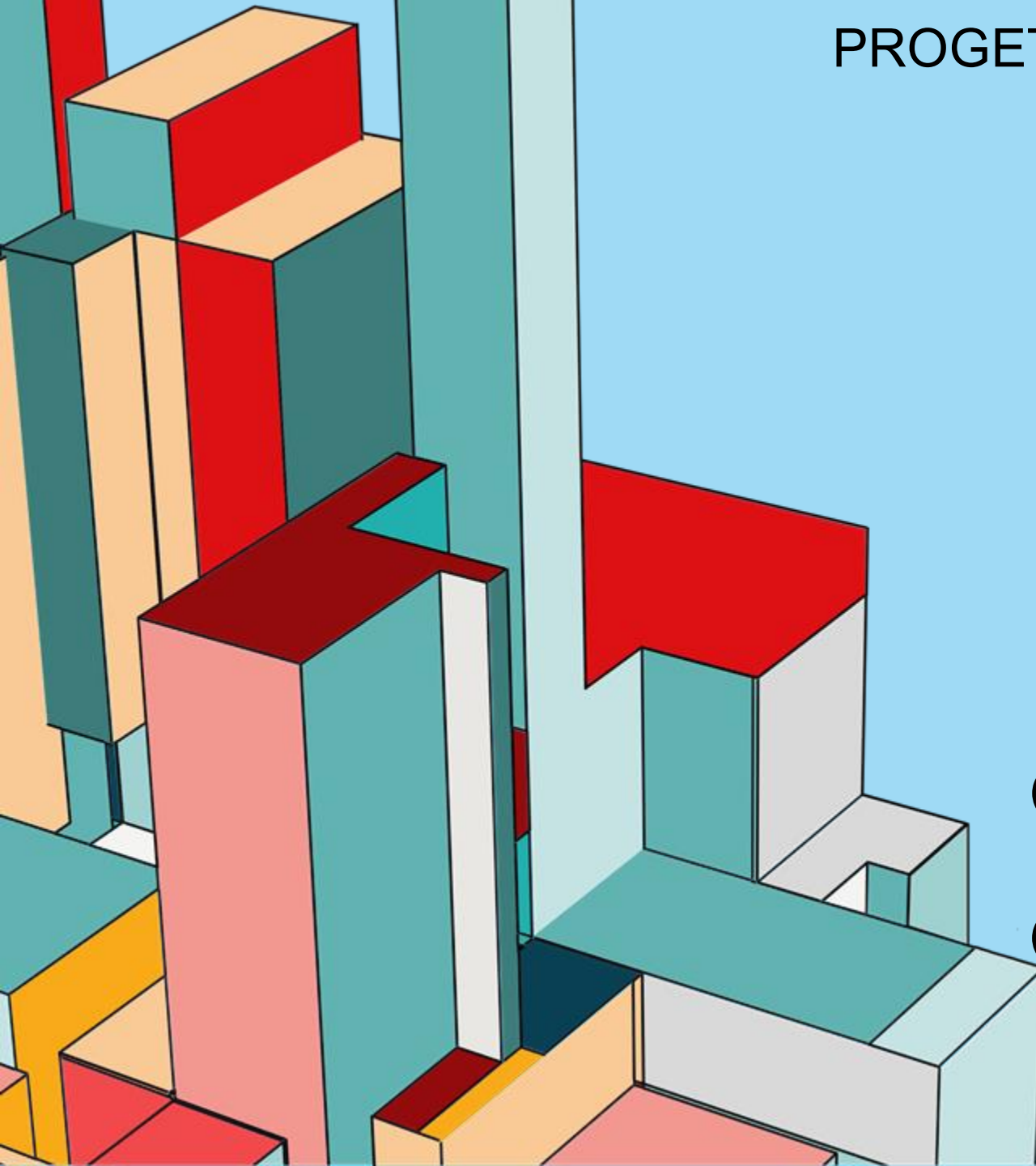


An abstract 3D geometric composition of various colored blocks (red, orange, teal, white, and dark blue) arranged in a complex, overlapping structure on the left side of the slide.

PROGETTO DI DISTRIBUTED SYSTEMS

A.A. 2024/2025

DUBITO ONLINE

ANDREA BIANCHI

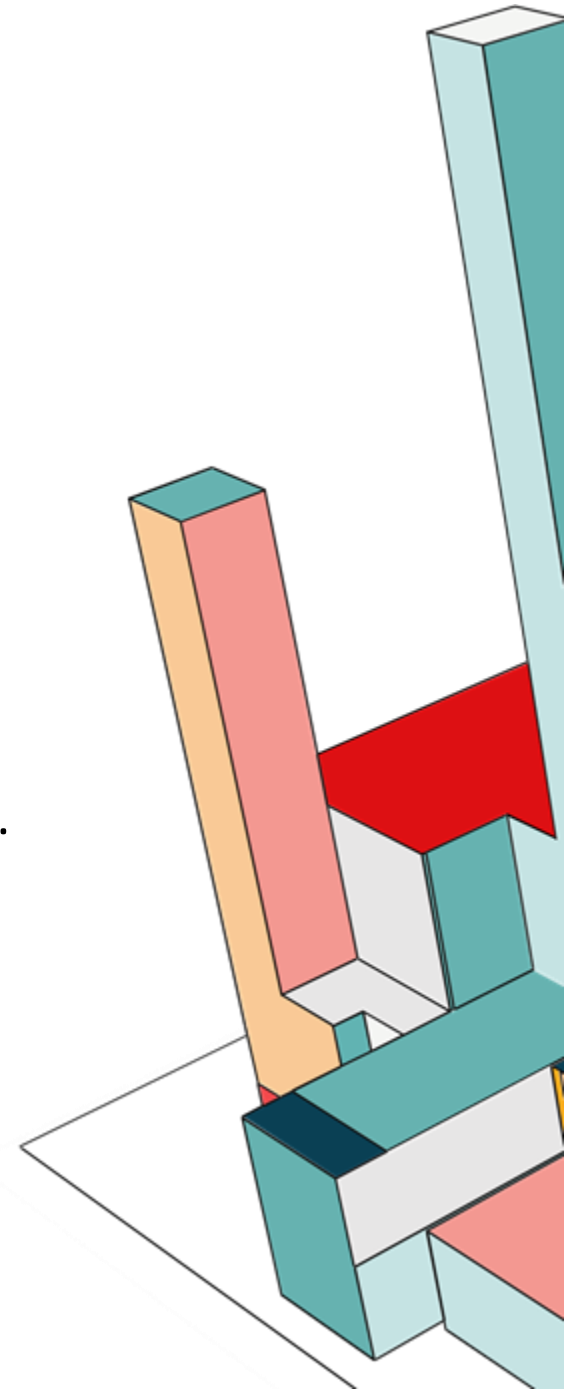
(ANDREA.BIANCHI24@STUDIO.UNIBO.IT)

ALBERTO ARDUINI

(ALBERTO.ARDUINI3@STUDIO.UNIBO.IT)

CHE COS'È DUBITO ONLINE

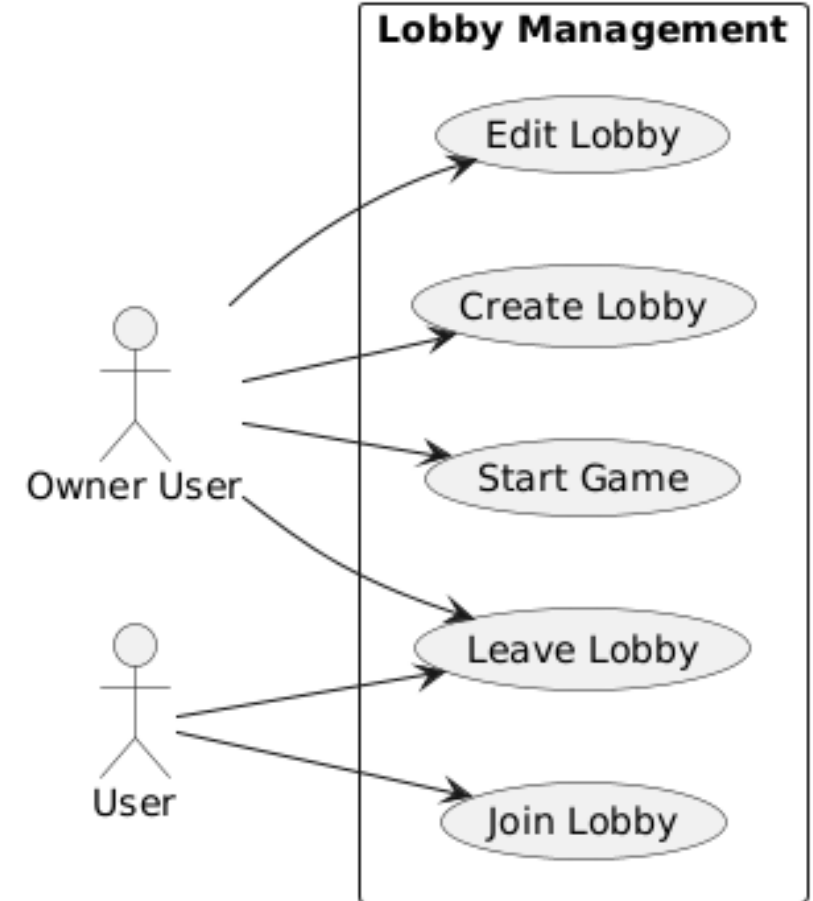
- Dubito Online è un'applicazione che permette agli utenti di unirsi a una partita online del gioco "Dubito", un noto gioco di carte incentrato sul mentire e confondere gli altri giocatori per poter vincere.
- Il sistema permette ad un utente di connettersi a un server, creare o unirsi a una lobby (una "stanza" in cui i giocatori possono trovarsi prima di iniziare una partita) e iniziare una nuova partita.
- Il gioco supporta da un minimo di 2 fino a un massimo di 4 giocatori per partita.



CASI D'USO

GESTIONE DELLE LOBBY

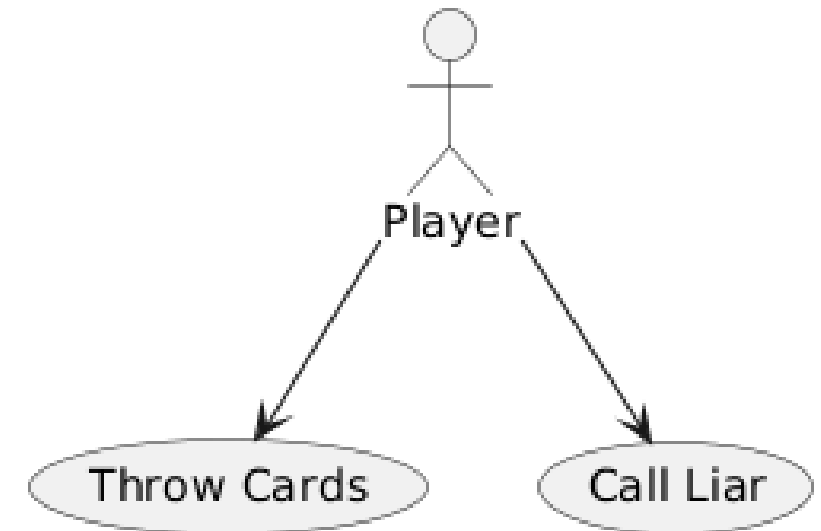
- L'utente owner crea una lobby;
- L'utente owner modifica la propria lobby;
- L'utente si unisce alla lobby;
- L'utente (owner o no) esce dalla lobby;
- L'utente owner fa cominciare il gioco.

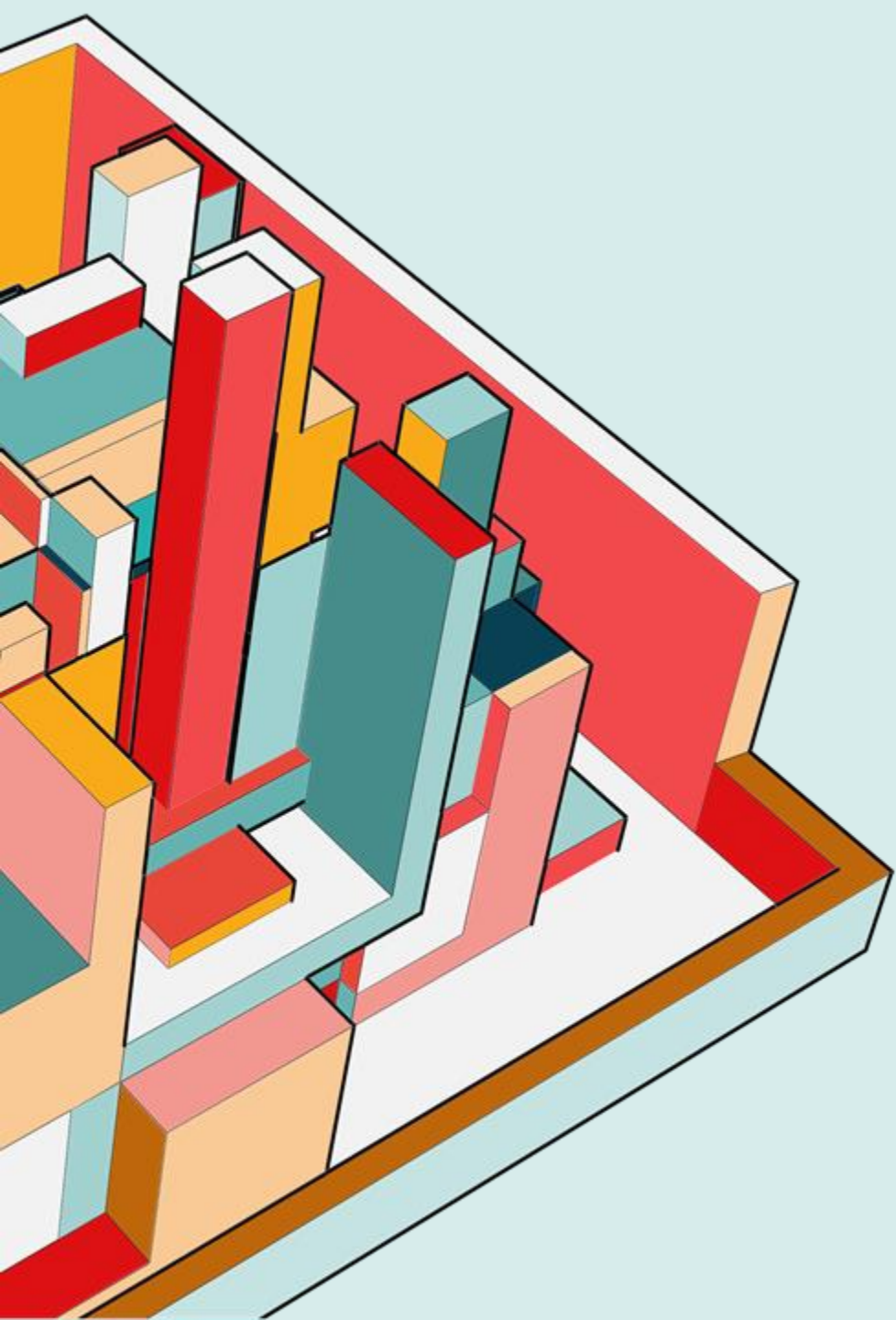


CASI D' USO

DURANTE LA PARTITA

- L'utente gioca delle carte dalla propria mano;
- L'utente chiama il giocatore precedente "Bugiardo";





ARCHITETTURA

Come si struttura il progetto

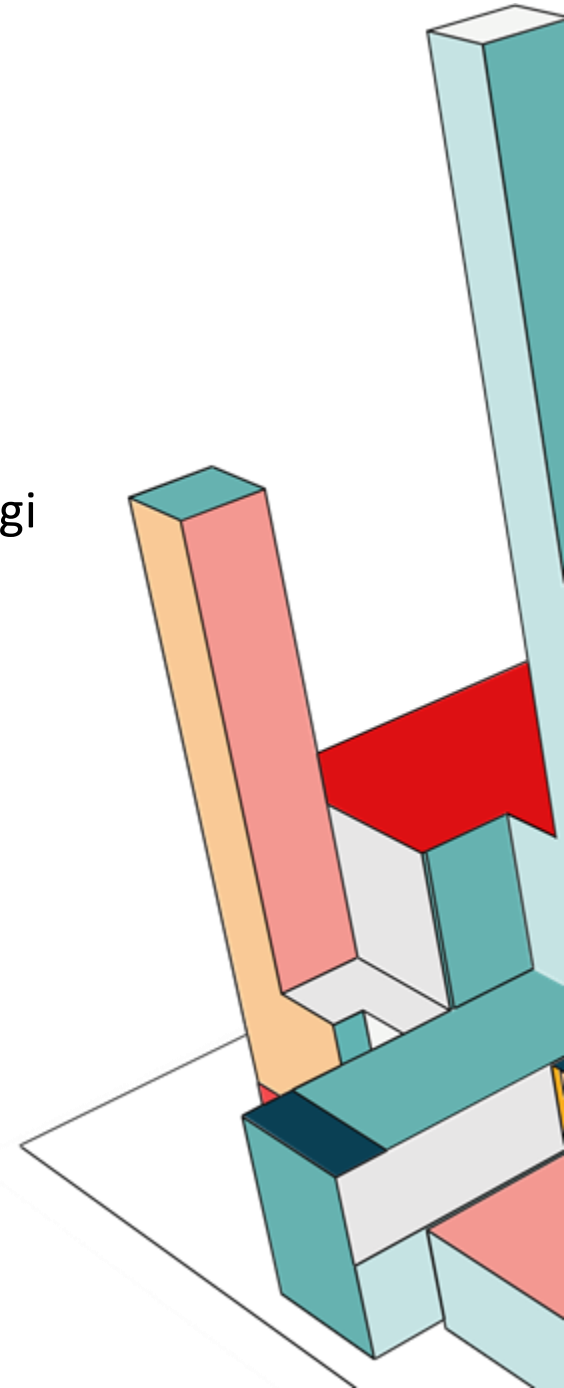
ARCHITETTURA

STRUTTURA DEL PROGRAMMA

- L'applicazione client utilizza l'architettura MVC per definire la logica e il programma;
- L'applicazione server è costruita tramite degli *handler* che intercettano i messaggi dei client, rispondendo con il loro stato aggiornato;

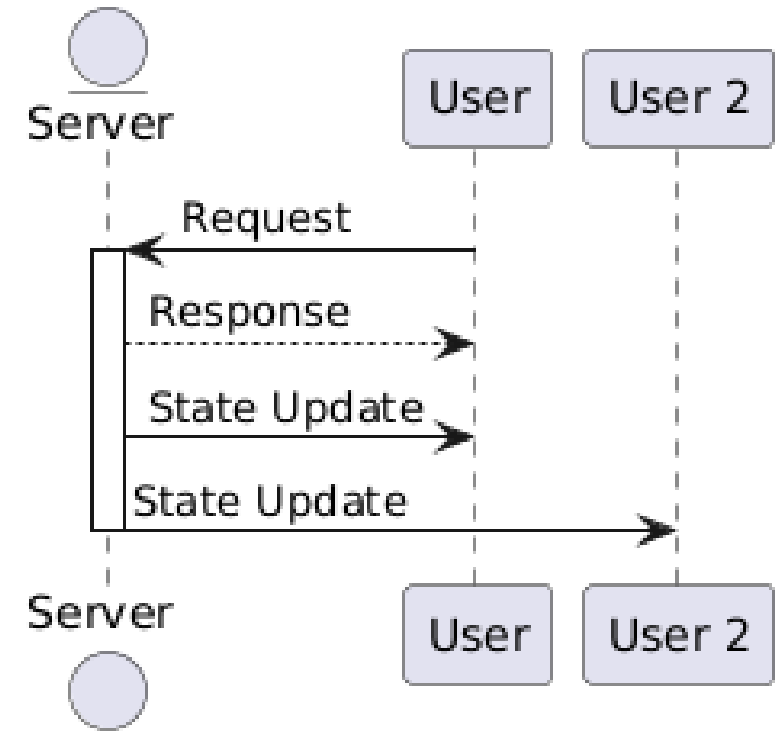
ARCHITETTURA DI RETE

- Lo scambio dei dati è fatto utilizzando JSON tramite TCP;
- Il sistema delle lobby è stato realizzato seguendo l'architettura **Client-Server**;
- Quando l'owner di una lobby richiede di cominciare la partita, viene instaurata una rete separata Peer-to-Peer (**P2P**) tra i giocatori;



ARCHITETTURA SERVER LOBBY

- Gli utenti si connettono a un server, situato sulla stessa macchina o su una macchina separata specificando il suo indirizzo IP e la porta;
- Una volta connessi, gli utenti possono effettuare richieste al server per gestire il proprio utente o effettuare operazioni sulle lobby (crearne, unirsi, etc);
- Il server elabora le richieste degli utenti e fornirà poi la risposta appropriata, inviando eventualmente il suo stato aggiornato agli altri utenti;



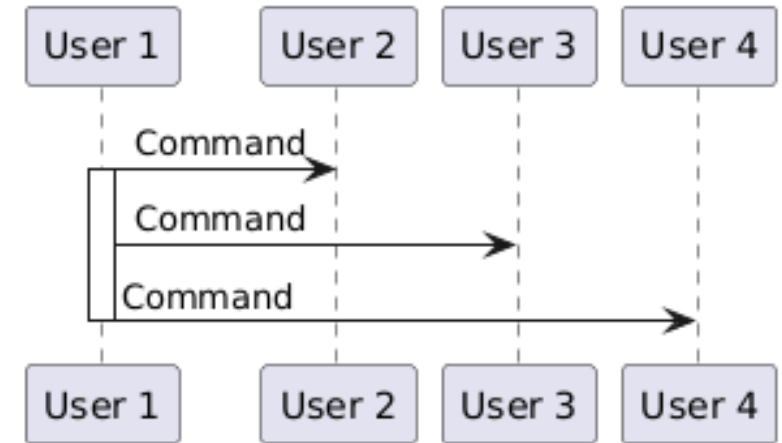
CONSISTENZA DEI DATI NEL SERVER LOBBY

La consistenza delle informazioni gestite dal server è raggiunta tramite un mix di svariati modelli di consistenza di tipo client-centric quali:

- Monotonic writes e reads, garantendo che le operazioni di scrittura del server vengano completate prima di qualunque operazione richiesta a seguire.
Inoltre, le operazioni di lettura dei dati del server restituiscono sempre le informazioni più aggiornate, tutti i clienti avranno quindi sempre una replica dello stato del server aggiornata;
- Read-your-writes, dove ogni scrittura effettuata sul server viene sempre seguita da un'operazione di lettura del suo stato da parte dei clienti connessi: questa operazione avrà sempre successo e permetterà la conferma ai clienti di avere informazioni correttamente aggiornate;
- Writes-follow-reads, che garantisce a ogni operazione di scrittura sul server di operare sulle informazioni lette precedentemente, e che queste corrispondano alle più aggiornate, garantendo così che ogni cliente abbia una copia dello stato del server aggiornata.

ARCHITETTURA DURANTE LA PARTITA

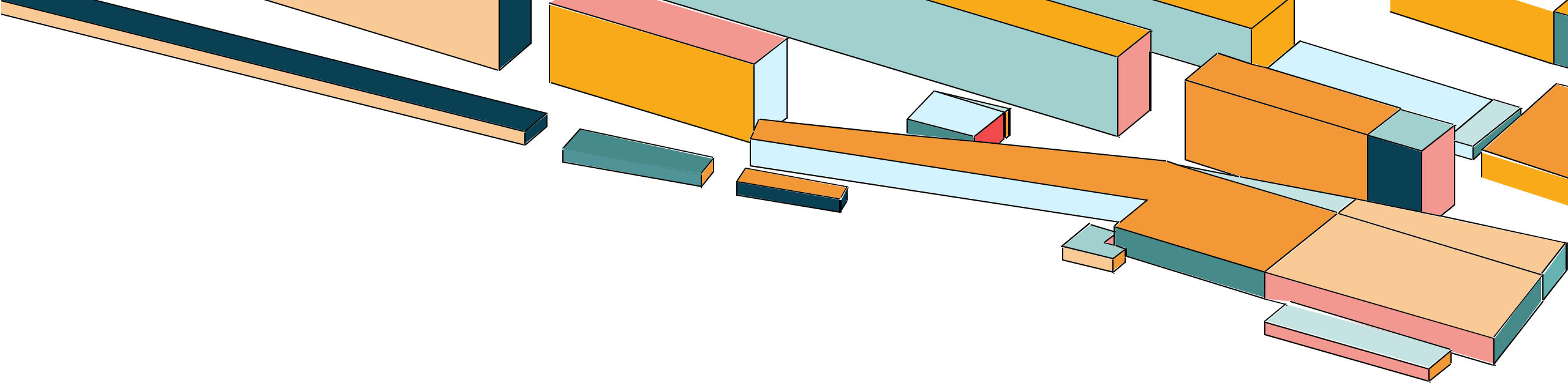
- Ogni utente assume il ruolo di "Peer", ed è in grado di comunicare con tutti gli altri Peer della rete;
- Quando un utente effettua un'azione, aggiorna il suo stato locale, per poi inviare un messaggio a tutti gli altri Peers, affinché essi aggiornino il loro stato locale;



- Il giocatore owner è colui che gestisce lo stato non specifico degli utenti, come l'impostazione della carta del round;
- Se l'owner si disconnette, l'ownership della partita passa al successivo giocatore vivo nell'ordine dei turni della partita;

REPLICAZIONE DELLO STATO DELLA PARTITA

- Lo stato del gioco durante una partita è replicato tra i peer della rete P2P tramite *state-machine replication*: infatti, ciò che viene inviato tra i peer sono le *azioni* dei giocatori, mai lo stato vero e proprio;
- Questo semplifica la gestione dello stato, in quanto possiamo sfruttare le operazioni già presenti di modifica dello stato in base ai messaggi in ingresso;
- Inoltre, ci permette di non avere un peer “master” che funge da fonte di verità per lo stato.



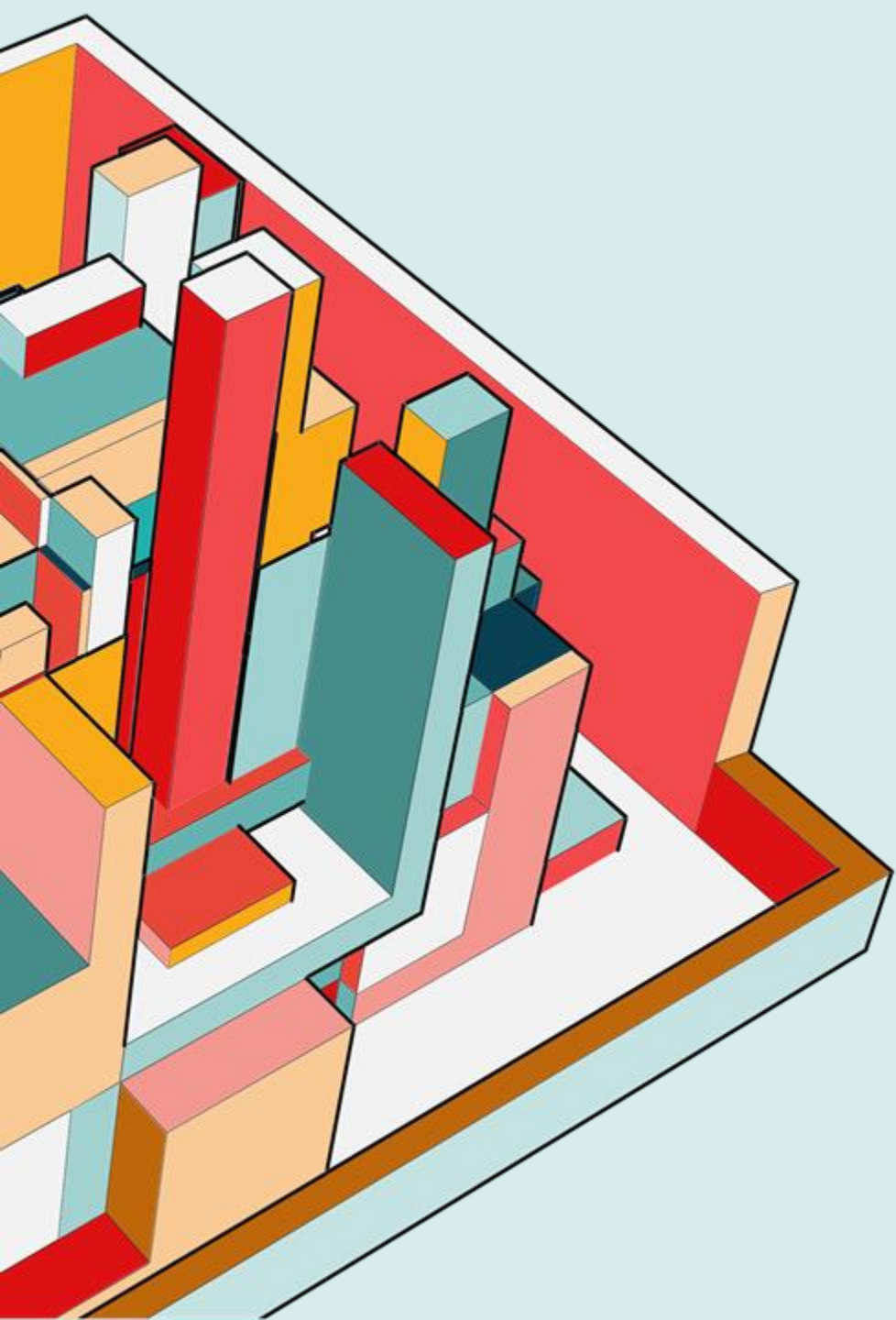
PERCHÉ QUESTE SCELTE?

QUALITY ATTRIBUTES

- I requisiti si incentrano più sul garantire *consistenza, correttezza e attendibilità*, rispetto che *disponibilità*;
- Abbiamo, quindi, impostato il sistema per:
 - Mantenere gli utenti sempre sincronizzati e in uno stato consistente, a costo di ridurre la disponibilità e a un leggero aumento della latenza;
 - Garantire il corretto funzionamento anche per lunghi lassi di tempo;

SCELTE ARCHITETTURALI

- Per quanto riguarda la gestione client-centric delle lobby, è stata scelta perché, seguendo i requisiti del sistema, non è necessario per gli utenti avere sempre la vista aggiornata del sistema, è richiesto solo che le loro operazioni siano coerenti con il loro stato locale.
- L'utilizzo della state-machine replication durante la partita, invece, deriva dai presupposti di limitare il traffico in rete e di poter riutilizzare il codice già scritto, facendo una mappatura tra gli eventi distribuiti e le interazioni che l'utente può avere con il sistema.



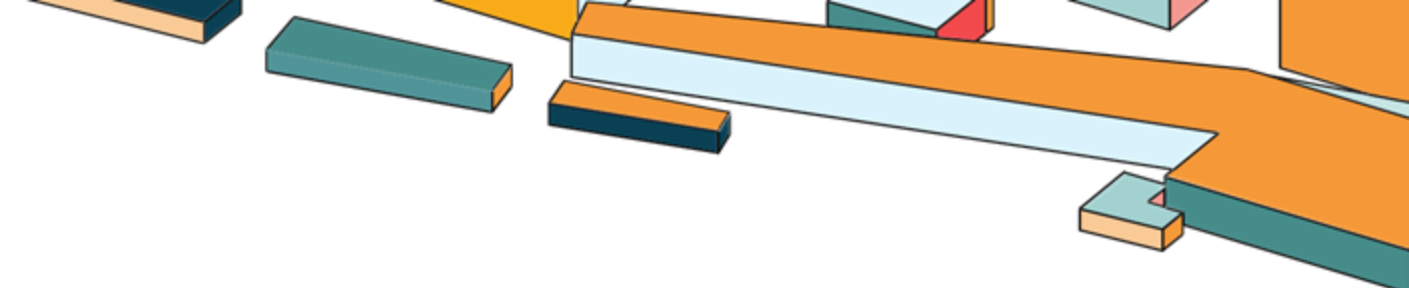
TESTING

Quali test sono stati fatti

UNIT E INTEGRATION TESTING

- Sono stati scritti vari *unit-test* per la verifica del corretto funzionamento dei componenti dell'applicazione:
 - a. Librerie di utilità;
 - b. Logica del gioco;
 - c. Corretta gestione dei messaggi (invio, ricezione, bufferizzazione e serializzazione);
 - d. Realizzazione/aggiornamenti/mantenimento delle lobby;
 - e. Gestione della rete e delle connessioni tra i peer.
- Sono anche stati stesi alcuni *integration-test* per la verifica delle parti critiche dell'architettura del sistema;
- Tutti i test sono stati scritti usando:
 - a. **JUnit 5 (Jupiter) come suite di test;**
 - b. **Mockito per il mock degli oggetti;**
 - c. **Una libreria interna per la creazione dei mock.**





TESTING MANUALE

- Abbiamo svolto una serie di prove manuali per verificare le azioni compiute tipicamente dagli utenti che usano l'applicativo;
- Per far ciò, abbiamo simulato l'utilizzo normale del sistema come una "Black box", avviando un server e una o più applicazioni client, verificando che il sistema si comportasse correttamente e che, anche in caso di errore, rimanesse consistente e responsivo;
- Abbiamo anche verificato il funzionamento del sistema in rete, utilizzando un sistema esterno (in particolare ngrok) per permettere il collegamento via NAT in internet;

GRAZIE PER LA VOSTRA ATTENZIONE

Andrea Bianchi

Alberto Arduini

