ANALISI DEI REQUISITI

Casi d'uso

L'**utente** deve poter:

- notificare tramite smartbell l'interesse di entrare nella tearoom
- essere accompagnato ad un tavolo libero dal waiter
- ordinare un tè comunicandolo al waiter
- consumare il tè
- comunicare al waiter di aver concluso la consumazione
- pagare al waiter con carta di credito
- lasciare la tearoom accompagnato dal waiter

Il **manager** deve poter :

- visualizzare lo stato corrente della tearoom tramite interfaccia web

Componenti del Sistema

La struttura del sistema consta di più componenti, che potenzialmente si trovano su nodi computazionali distinti in quanto si tratta di dispositivi elettronici diversi.
I componenti corrispondono alle entità presenti nel problema, che vengono così individuate: **waiter**, **smartbell** e **barman**. Queste entità infatti necessitano la definizione di un comportamento, in modo da poter soddisfare richieste che arrivano dall'esterno del sistema.
Le richieste in questione possono arrivare da un cliente o dal manager della tearoom.
Perché queste entità possano costituire un sistema è necessario che comunichino tra loro.
La comunicazione deve poter avvenire in modo sincrono o asincrono in quanto determinate azioni devono poter essere svolte in sequenza o in parallelo.
Trattandosi di un sistema distribuito diventa necessario l'utilizzo di messaggi.

Scelta del Paradigma

La rappresentazione dei componenti può avvenire secondo diverse modalità.
Un primo approccio potrebbe essere quello a Oggetti Java.
Gli oggetti sono caratterizzati da una rappresentazione dello stato tramite campi interni; definizione delle funzionalità tramite funzioni e procedure; interazione tramite *procedure call*.
Nascono per eseguire su un solo nodo computazionale.
Gli oggetti hanno quindi delle limitazioni se utilizzati in ambito distribuito.
Alcune di queste sono: l'assenza di flusso di controllo indipendente e soprattutto l'impossibilità di eseguire *procedure call* in remoto.

Queste caratteristiche si possono trovare cambiando paradigma in favore di uno ad Attori.
Gli attori infatti hanno un flusso di controllo indipendente e comunicano tramite messaggi *by design* su nodi diversi.

Per un confronto con altre tecnologie riferirsi al file: "Confronto tecnologie"

Per questo è stato analizzato il linguaggio di modellazione Qactor.
Date le sue caratteristiche è quello più adatto a gestire questo problema.
Il linguaggio prevede degli Attori Qactor come entità principali capaci di comunicare in un ambiente distribuito tramite scambio di messaggi.
Questi messaggi permettono una semantica sia sincrona (Request/Reply) che asincrona (Dispatch/Event).

Inoltre trattandosi di un linguaggio di modellazione, permette di definire le entità in gioco senza legarsi ad una specifica tecnologia. Ad esempio in questo caso il Qactor genererà del codice Kotlin in automatico, ma si potrebbe personalizzare diversamente.
Da ora, sarà utilizzato il linguaggio Qactor per formalizzare gli agenti e i messaggi.

Test plan
All'interno del test saranno verificati i seguenti punti:

- garantire l'accesso al client soltanto se la sua temperatura è inferiore a 37.5°C e se c'è almeno un tavolo pulito. Verifica che il client riceva un Client ID dalla smartbell;
- verificare che il client arrivi ad un tavolo pulito (accompagnato dal waiter);
- verificare che il waiter consegni il tè giusto al tavolo da cui è partito l'ordine;
- verificare che il waiter ottenga il pagamento se scade il tempo massimo per la consumazione del client (maxstaytime);
- verificare che il waiter accompagni il client alla exitdoor.

Il testing dell'applicazione sarà effettuato tramite JUnit.
Il testing con JUnit permette di automatizzare parte dei comportamenti e di verificare che i risultati siano coerenti con le aspettative.

Approccio incrementale
Lo sviluppo del sistema procederà in maniera incrementale, partendo da problemi semplificati con vincoli e assunzioni, in modo da velocizzare il rilascio di prototipi.
Nello specifico il lavoro verrà organizzato secondo il metodo Scrum, eseguendo sprint e review.

Per il primo sprint, verranno considerati i seguenti vincoli:

- nel sistema c'è soltanto un cliente
- il cliente esegue le operazioni correttamente
- un nuovo cliente non chiede mai di entrare prima che sia uscito l'altro