13/07/2020

**PIAZZA\_SPRINT 4**

## Sprint Goal

Essere in grado di visualizzare la current situation della tearoom nella Web GUI.

## Requisiti

-I requisiti sono gli stessi elencati nel file [TFBO20ISS.pdf](../../../../DOCUMENTS/TFBO20ISS.pdf) con **le stesse assunzioni fatte nello Sprint 3**.

-Assunzioni:

1) Si rilassa il vincolo de "un solo cliente in sala". Ora possono arrivare tutte le richieste di ingresso che si vuole. Come da requisiti, la stanza potrà accogliere fino a N=2 clienti contemporaneamente.

2) Il barman **non** è in grado di lavorare in parallelo a più ordini. Le preparazioni, supposte di breve durata, sono sequenziali.

3) Il tempo di preparazione di un ordine è sempre lo stesso, a prescindere da cosa è stato ordinato.

4) I task del waiter **non** sono interrompibili.

## Analisi dei Requisiti

Il requisito da soddisfare è il seguente:

As a ***manager***:

I intend to be able to see the ***current state*** of the ***tearoom*** by using a browser connected to a web-server associated to the application.

### Bozze di Test Plan

Pensare e descrivere QUI un qualche scenario d’esecuzione particolare che desidero poi tradurre in un testPlan finale da mostrare al committente nella parte di video in cui faccio vedere una simulazione automatizzata.

## Analisi del Problema

### Problematiche

Per rendere più proficuo il momento di confronto e feedback con il committente era già emersa nello SPRINT 3 la necessità di realizzare una Web GUI in cui fosse possibile simulare il comportamento e l’interazione dei clienti.

Per soddisfare il requisito della *current situation* è possibile utilizzare la stessa web GUI sviluppata nello Sprint precedente, arricchendola dei dati necessari.

Problematica(fase di Analisi del Problema): Devo far visualizzare lo stato corrente della stanza sulla webGUI. Dal momento che le informazioni relative allo stato della stanza si trovano lato server (nel progetto it.unibo.iss.sprintN) e io devo visualizzarle nella GUI alcune sotto-problematiche che nascono sono:

-il problema richiede che le informazioni vengano trasmesse dal server alla GUI:

Deve avere l’iniziativa la webGUI facendo richiesta dello stato del server, oppure è il server a dover prendere l’iniziativa e inviare le informazioni di stato alla webGUI?

-devo chiedermi quando è opportuno trasmetterle.

Dal momento che da requisiti si chiede di visualizzare la current situation sarà necessario mandare il nuovo stato ogni volta che questo cambia in modo da poter visualizzare sulla GUI lo stato sempre aggiornato.

SOLUZIONE (Progetto): (dall’analisi: “è necessario mandare il nuovo stato ogni volta che questo cambia in modo da poter visualizzare”) Alla luce di ciò e chiaro come né una soluzione in cui la webGUI interroga a polling il server, né una soluzione in cui il server manda periodicamente informazioni alla webGUI siano adeguate.

Una soluzione basata sul pattern Observer è molto più adatta a risolvere questa problematica: dichiariamo lo stato della stanza come Observable. A questo punto sarà possibile agganciarle uno o più observer che recepiscano il cambiamento di stato e agiscano di conseguenza (facciano visualizzare lo stato sulla pagina).

Ulteriore vantaggio del pattern observer: L’entità observable non sa a priori chi e quanti saranno gli observer che sottoscriveranno il proprio interesse (maggior disaccoppiamento tra le entità).

Nascono 2 nuove problematiche:

-observer locali oppure observer remoti? Possiamo sfruttare il fatto che l’infrastruttura renda i QActor delle entità osservabili anche in remoto tramite protocollo Coap. Nel nostro caso lo stato si trova nel waiter a cui basterà fare un updateResource[# “$nuovoStato” #] ogni volta che lo stato cambia. Potremmo così utilizzare un observer remoto che si trovi sulla pagina Web il quale, ogni volta che recepisce un cambiamento di stato, lo mostra sulla pagina Web.

-single page application con webSocket oppure interazione REST ?

Stato della tearoom? Centralizzato o decentralizzato? Lo stato corrente della stanza è bene che sia centralizzato sia per far fronte al requisito del manager sia per facilitare il testing che quindi può ruotare tutto intorno ad esso.

Lo stato corrente può essere visto come stato dei tavoli e stato del waiter…. Altre informazioni vengono demandate a sviluppi futuri, se richieste esplicitamente dal committente.

È il waiter che cambia lo stato in seguito ad una propria azione? Oppure è lo stato che cambia e in base al proprio cambiamento comanda le azioni del waiter (Visione domain centered-Modello Esagonale? Guardala per la discussione finale)

In linea teorica cosa comporterebbe se pensassi ad un refactoring del codice per applicare il modello esagonale e tutto ruota attorno ad un dominio ? (Che nel nostro caso sarà lo stato della stanza).

Provare a pensare alla tematica e magari la discuteremo all’orale.

### Architettura Logica

### Modello eseguibile

## Test Plan

## Progetto

Alcune note:

Nell’immagine di progetto far vedere con delle frecce anche l’interazione CoapObserver 🡨🡪 QActorObservable.

**SPRINT 4 – REVIEW**

Il requisito opzionale one client in the hall comporta che, una volta fatta la ring della smartbell, la successiva non potrà essere fatta fino a quando il waiter non ha fisicamente accolto chi ha suonato.

Una ring non può essere mandata se il waiter sta eseguendo un task convoyToExit. Viceversa un convoyToExit non può essere fatto se è stata fatta una ring poiché significa che la hall è occupata da chi ha suonato.

Quindi queste due azioni diventano due azioni che per eseguire devono richiedere un lock sulla risorsa hall, la quale andrà modellata come una risorsa mutualmente esclusiva.

**SPRINT 4 – RETROSPECTIVE**

Towards a Exagonal Architecture:

L’idea per un refactoring future può essere quella di andare verso un’architettura della nostra applicazione che non sia più layered come è stata impostata fino ad ora, bensì **esagonale(LINK).**

Dallo studio dell’applicazione infatti emerge come il domain Model sia centrale e determinante in ogni scelta che viene presa da waiter.

Possiamo pensare quindi di mettere al centro il domain model e, quando questo riceve degli eventi che ne inducono un cambiamento (es: messaggi con richieste di ingresso) potrà invocare determinate azioni sul waiter (che in questo modo diventa un vero e proprio servizio).

Ora le azioni del waiter sono subordinate ad un’invocazione richiesta dal domain Application che tenta di cambiare il proprio stato: trovandoci ad eseguire in un *Real World* se poi il servizio invocato andrà a buon fine lo stato cambierà altrimenti no.

Ora non è più il waiter che fa le azioni e comanda il cambiamento del domain Application. Si è ribaltata la visione.

PER LA REVIEW FINALE:

-Fare un video demo in cui uso la web GUI e mostro l’interazione desiderata con il waiter.

Poi a fine video dire: “la stessa cosa che ho mostrato adesso con un’interazione “a mano” può essere automatizzata da dei testPlan che simulino esattamente determinati scenari.

Ne vediamo uno insieme in cui il robot farà

-questo questo e quello perché ci troviamo in questo determinato scenario….