



>Three Way Milkshake_

Verbale esterno 6

23 Marzo 2021

Three Way Milkshake - Progetto "PORTACS"

threewaymilkshake@gmail.com

Versione	1.0.0
Stato	Approvato
Uso	Esterno
Approvazione	Zuccolo Giada
Redazione	De Renzis Simone
Verifica	Greggio Nicolò
Destinatari	Three Way Milkshake Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Verbale del meeting del 2021-03-23 del gruppo Three Way Milkshake con il proponente Sanmarco Informatica.



Registro delle modifiche

Vers.	Descrizione	Data appr.	Approvazione
1.0.0	Approvazione del verbale	2021-03-28	Zuccolo Giada

Vers.	Descrizione	Redazione	Data red.	Verifica	Data ver.
0.1.0	Stesura del verbale	De Renzis Simone	2021-03-26	Greggio Nicolò	2021-03-28



Indice

1	Informazioni generali	3
1.1	Dettagli sull'incontro	3
1.2	Ordine del giorno	3
2	Verbale della riunione	4
2.1	Organizzazione dell'architettura del software	4
2.1.1	Server	4
2.1.2	Client	4
2.2	Analisi dei design pattern da adottare	4
2.3	Guida manuale	5
3	Tracciamento temi affrontati	6



1 Informazioni generali

1.1 Dettagli sull'incontro

- **Luogo:** Incontro telematico tramite piattaforma Google Meet;
- **Data:** 2021-03-23;
- **Ora di inizio:** 17:00;
- **Ora di fine:** 18:00;
- **Partecipanti interni:** (6/6)
 - Chiarello Sofia;
 - Crivellari Alberto;
 - De Renzis Simone;
 - Greggio Nicolò;
 - Tessari Andrea;
 - Zuccolo Giada.
- **Partecipanti esterni:** (1)
 - Beggiato Alex (Sanmarco Informatica).

1.2 Ordine del giorno

La riunione prevede la discussione con il proponente dei seguenti punti:

- organizzazione dell'architettura del software;
- analisi dei design pattern da adottare;
- guida manuale.



2 Verbale della riunione

2.1 Organizzazione dell'architettura del software

È stata intavolato un confronto con il proponente per definire alcuni dettagli riguardanti l'architettura del software.

2.1.1 Server

Per quanto riguarda la componente server, sono stati definiti 3 livelli su cui si può articolare l'architettura:

1. **layer di comunicazione:** gestisce la comunicazione tra il server e i client, e si può specializzare in due sezioni: la prima può essere gestita tramite Socket, e riguarderà l'invio dei dati per il monitor real-time (di visualizzazione del magazzino con i muletti che si muovono al suo interno); la seconda, gestibile tramite API di tipo REST, regola l'interazione dei due attori "Amministratore" e "Responsabile" con l'interfaccia grafica. Per approcciarsi a questa pratica, il proponente suggerisce di usufruire della libreria "Jersey" di Java, indicando come possibile fonte di studio una guida dedicata sul sito html.it;
2. **layer di business:** il motore di calcolo del sistema;
3. **layer di persistenza:** per la gestione della persistenza, nel caso del nostro applicativo per il momento è previsto il salvataggio in file *.json*. È stato evidenziato come sia importante assicurare, a questo livello, la maggior indipendenza possibile dagli altri layer per consentire l'estensione ad altri tipi di persistenza.

2.1.2 Client

I client si differenziano a seconda della tipologia, entrambi si compongono di una parte node e una angular:

- muletti:
 - node: comunicazione con il server;
 - angular: interfaccia di visualizzazione propria mappa e comandi di guida;
- utenti (admin e responsabile):
 - node: comunicazione mappa real time;
 - rest (da valutare): per le altre azioni (aggiunta/modifica/rimozione utenti/task ...).
 - angular: interfaccia di login, visualizzazione e modifica dati e varie funzionalità in base ai privilegi.

2.2 Analisi dei design pattern da adottare

Sono state evidenziate alcune componenti del server che potrebbero essere modellate tramite il design pattern *Singleton*:

1. path finder: che incapsula l'algoritmo per la ricerca del percorso migliore;



2. collision detection: per la gestione delle collisioni;
3. warehouse map: la rappresentazione interna della mappa del magazzino.

Per queste classi è necessario assicurare la presenza di un'unica istanza condivisa, motivo per cui il pattern Singleton può essere impiegato efficacemente.

Per quanto riguarda ConnectionAcceptor (classe dedicata ad accettare le connessioni entranti) è stata pensata la possibilità di introdurre un pattern di tipo *Factory*.

2.3 Guida manuale

È stata discussa l'implementazione della guida manuale per i muletti: in particolare è stato evidenziato come non debba esserci alcuna attesa di feedback da parte del server nel movimento azionato da guida manuale. Tuttavia è comunque necessario, per le unità guidate manualmente, la comunicazione con il server della propria posizione, necessarie per il calcolo (ed eventualmente, i ricalcoli) del percorso migliore per raggiungere la destinazione. Una possibilità implementativa per assicurare queste caratteristiche è l'introduzione di un timer interno al client che regoli lo spostamento dell'unità sulla base degli input utente ricevuti.

3 Tracciamento temi affrontati

Codice	Decisione
VE_6.1	Architettura del server a 3 layer.
VE_6.2	Comunicazioni con i Muletti avvengono tramite TCPSocket (richiedono componente in Node.js).
VE_6.3	Comunicazioni con utenti potrebbero avvenire tramite REST API per le operazioni varie e tramite TCPSocket per la ricezione dei dati della mappa globale.
VE_6.4	Path finder, collision detection e warehouse map possono beneficiare del design pattern Singleton.
VE_6.5	ConnectionAcceptor può beneficiare del design pattern Factory.
VE_6.6	Possibilità di introdurre un timer interno ai client per temporizzare gli spostamenti in guida manuale.