



>Three Way Milkshake_

Manuale Manutentore

Three Way Milkshake - Progetto "PORTACS"

threewaymilkshake@gmail.com

Versione	2.0.0
Stato	Approvato
Uso	Esterno
Approvazione	Greggio Nicolò
Redazione	Three Way Milkshake
Verifica	Three Way Milkshake
Destinatari	Sanmarco Informatica Prof. Vardanega Tullio Prof. Cardin Riccardo Three Way Milkshake

Descrizione

Manuale di supporto allo sviluppo e manutenzione del software G
PORTACS

Registro delle modifiche

Vers.	Descrizione	Data	Approvazione
2.0.0	Approvazione documento	2021-05-23	Greggio Nicolò

Vers.	Descrizione	Redazione	Data	Verifica	Data
1.4.1	Modifica simbolo sezioni changelog	Tessari Andrea	2021-05-21	Crivellari Alberto	2021-05-21
1.4.0	Incremento §5.1.2.3	De Renzis Simone	2021-05-20	Crivellari Alberto	2021-05-20
1.3.0	Incremento §3	Greggio Nicolò	2021-05-18	Crivellari Alberto	2021-05-20
1.2.0	Incremento §5	Greggio Nicolò	2021-05-17	Crivellari Alberto	2021-05-20
1.1.1	Incremento §3.1.1	Tessari Andrea	2021-05-11	Chiarello Sofia	2021-05-12
1.1.0	Incremento §2.1.1	Zuccolo Giada	2021-05-11	Chiarello Sofia	2021-05-12

Vers.	Descrizione	Data	Approvazione
1.0.0	Approvazione documento	2021-04-27	Greggio Nicolò

Vers.	Descrizione	Redazione	Data	Verifica	Data
0.7.0	Stesura §6.1, §6.2.1 e §6.3	Crivellari Alberto	2021-04-25	Greggio Nicolò	2021-04-26
0.6.1	Stesura §5.1	De Renzis Simone	2021-04-24	Greggio Nicolò	2021-04-24
0.6.0	Stesura §5.2, §6.4 e §2.1	De Renzis Simone	2021-04-24	Greggio Nicolò	2021-04-25
0.5.2	Stesura §5.3.1	Crivellari Alberto	2021-04-24	De Renzis Simone	2021-04-25
0.5.1	Stesura §5.3.2	Greggio Nicolò	2021-04-23	De Renzis Simone	2021-04-25
0.5.0	Stesura §5.3 (introduzione) e §5.3.2	Greggio Nicolò	2021-04-23	De Renzis Simone	2021-04-25
0.4.0	Stesura introduzione §6 e §6.2.2	Tessari Andrea	2021-04-23	Zuccolo Giada	2021-04-23



0.3.1	Stesura §4	Crivellari Alberto	2021-04-23	De Renzis Simone	2021-04-25
0.3.0	Stesura §3	Tessari Andrea	2021-04-23	Greggio Nicolò	2021-04-26
0.2.1	Incremento §5.2	Zuccolo Giada	2021-04-22	Tessari Andrea	2021-04-22
0.2.0	Stesura §5.2	Tessari Andrea	2021-04-22	Zuccolo Giada	2021-04-22
0.1.2	Stesura §2.3	Greggio Nicolò	2021-04-22	Crivellari Alberto	2021-04-27
0.1.1	Stesura §2.2	Chiarello Sofia	2021-04-22	Tessari Andrea	2021-04-22
0.1.0	Stesura §1	Tessari Andrea	2021-04-22	Chiarello Sofia	2021-04-22
0.0.1	Impaginazione	De Renzis Simone	2021-04-15	Tessari Andrea	2021-04-16



Indice

1	Introduzione	7
1.1	Scopo del documento	7
1.2	Scopo del prodotto	7
1.3	Riferimenti	7
1.3.1	Normativi	7
1.3.2	Informativi	8
2	Tecnologie e librerie	9
2.1	Server	9
2.1.1	Tecnologie	9
2.1.2	Librerie e Framework	10
2.2	Client	11
2.2.1	Tecnologie	11
2.2.2	Librerie e Framework	11
2.3	Version Control System e Continuous Integration	12
2.3.1	Git e gitflow	12
2.3.2	GitHub	12
2.3.3	GitHub Actions	12
3	Setup	13
3.1	Requisiti di sistema	13
3.1.1	Requisiti Hardware	13
3.1.2	Requisiti Software	13
3.1.2.1	Esecuzione	13
3.1.2.2	Sviluppo	14
3.2	Installazione ed avvio degli applicativi	14
3.2.1	Server	14
3.2.2	Client	14
3.2.2.1	Terminazione dei container	15
3.3	Costruzione degli artefatti e delle immagini	16
3.3.1	Server	16
3.3.2	Client	16
3.3.3	Caricamento nel docker hub	16
4	Testing	17
4.1	Coveralls	17
4.2	GitHub Actions	17
4.3	Testing lato server	17
4.3.1	Junit	17
4.4	Testing lato client	17
4.4.1	Jasmine	17
4.4.2	Karma	17
4.4.3	Mocha	17



5	Architettura del sistema	18
5.1	Server	19
5.1.1	Communication layer	20
5.1.1.1	Incrementare il livello di sicurezza	20
5.1.2	Business layer	21
5.1.2.1	Clients	22
5.1.2.1.1	Introdurre nuove tipologie di utenti	23
5.1.2.2	Mappa	24
5.1.2.2.1	Estendere il path finding	25
5.1.2.3	Collisioni	26
5.1.2.3.1	Modificare handler per la gestione delle collisioni	27
5.1.3	Persistence layer	27
5.1.3.1	Implementare tipi di persistenza alternativi	28
5.2	Client	29
5.2.1	Muletti	29
5.2.2	Utenti (amministratori/responsabili)	30
5.2.2.1	Introdurre nuovi tipi di utenti	31
5.3	Comunicazione	32
5.3.1	Diagrammi di sequenza	32
5.3.2	Protocollo di comunicazione	34
5.3.2.1	Modificare il protocollo	34
5.3.2.1.1	Esempio modifica lato server	35
5.3.2.1.2	Esempio modifica lato client	35
5.3.2.2	Connessione: identificazione e login	35
5.3.2.2.1	Esempio connessione ed autenticazione muletto	35
5.3.2.3	Enumerazioni	36
5.3.2.4	Comandi clients → server	37
5.3.2.5	Comandi server → clients	40



Elenco delle figure

5.0.1	Rappresentazione ad alto livello dell'architettura del software	18
5.1.1	Visione complessiva dell'architettura del server	19
5.1.2	Visione di dettaglio del Communication Layer	20
5.1.3	Visione di dettaglio del package Clients	22
5.1.4	Visione di dettaglio del package Map	24
5.1.5	Visione di dettaglio del package Collision	26
5.1.6	Visione di dettaglio del Persistence Layer	27
5.2.1	Diagramma UML _A delle classi per l'unità	29
5.2.2	Diagramma UML _A delle classi per gli admin e i manager	30
5.3.1	Visione complessiva della connessione server Java - unità muletto in NodeJS	32
5.3.2	Esempio di comunicazione tra server e client, attraverso i socket TCP . . .	33



Elenco delle tabelle

5.3.1	Modificare il protocollo do comunicazione	35
5.3.2	Riepilogo enumerazioni	36
5.3.3	Comandi clients → server Forklifts	37
5.3.4	Comandi clients → server User generico	37
5.3.5	Comandi clients → server User Manager (Responsabile)	38
5.3.6	Comandi clients → server User Admin (Amministratore)	39
5.3.7	Comandi server → clients generici per tutti i client	40
5.3.8	Comandi server → clients Forklifts (muletti)	41
5.3.9	Comandi server → clients User generico	42
5.3.10	Comandi server → clients Utente Manager (responsabile)	42
5.3.11	Comandi server → clients Utente Admin (amministratore)	43



1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è presentare tutte le informazioni necessarie al mantenimento e all'estensione del software PORTACS_A, mostrando nel dettaglio l'architettura del sistema e l'organizzazione del codice sorgente.

In questo documento saranno presentate le varie tecnologie usate, sia lato front end che back end, come anche le varie librerie e framework_G. Verrà inoltre mostrato il sistema di versionamento utilizzato e la Continuous Integration applicata.

1.2 Scopo del prodotto

Il capitolato_G C5 propone un progetto_G in cui viene richiesto lo sviluppo di un software per il monitoraggio in tempo reale di unità che si muovono in uno spazio definito. All'interno di questo spazio, creato dall'utente per riprodurre le caratteristiche di un ambiente reale, le unità dovranno essere in grado di circolare in autonomia, o sotto il controllo dell'utente, per raggiungere dei punti di interesse posti nella mappa. La circolazione è sottoposta a vincoli di viabilità e ad ostacoli propri della topologia dell'ambiente, deve evitare le collisioni con le altre unità e prevedere la gestione di situazioni critiche nel traffico.

1.3 Riferimenti

1.3.1 Normativi

- Model-View Patterns:
<https://medium.com/@anshul.vyas380/mvc-pattern-3b5366e60ce4>;
- SOLID Principles:
https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design;
- Diagrammi delle classi:
https://www.math.unipd.it/~rcardin/swea/2021/DiagrammidelleClassi_4x4.pdf;
- Diagrammi dei package:
https://www.math.unipd.it/~rcardin/swea/2021/DiagrammideiPackage_4x4.pdf;
- Diagrammi di sequenza:
https://www.math.unipd.it/~rcardin/swea/2021/Diagrammidisequenza_4x4.pdf;
- Design Pattern Creazionali:
<https://refactoring.guru/design-patterns/creational-patterns>;
- Design Pattern Strutturali:
<https://refactoring.guru/design-patterns/structural-patterns>;
- Design Pattern Comportamentali:
<https://refactoring.guru/design-patterns/behavioral-patterns>.



1.3.2 Informativi

- **GLOSSARIO**: per la definizione dei termini (pedice G) e degli acronimi (pedice A) evidenziati nel documento;
- Capitolato d'appalto C5-PORTACS:
<https://www.math.unipd.it/~tullio/IS-1/2020/Progetto/C5.pdf>
- Software Engineering - Iam Sommerville - 10th Edition.
- Angular:
<https://angular.io/>;
- Node.js:
<https://nodejs.org/en/>;
- PrimeNG:
<https://www.primefaces.org/primeng/>;
- Java:
<https://www.java.com/it/>;
- Spring:
<https://spring.io/>
<https://start.spring.io/>;
- Docker:
<https://www.docker.com/>
<https://dockertutorial.it/>.



2 Tecnologie e librerie

Nelle sezioni che seguono, vengono elencate le tecnologie e le librerie interessate dallo sviluppo del software. Per ognuna, viene presentata una breve descrizione e spiegato il suo impiego nel contesto del software. Dove necessario, viene fornito un collegamento per il download e l'installazione delle risorse necessarie per lo sviluppo e manutenzione del progetto.

2.1 Server

2.1.1 Tecnologie

- **Java:** linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, si appoggia sull'omonima piattaforma software di esecuzione (Java Virtual machine), specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione. La componente server del software è realizzata interamente con questo linguaggio.

- Versione utilizzata: JavaSE-15
- Documentazione: <https://docs.oracle.com/en/java/javase/15/>

Per il download e l'installazione si rimanda alla documentazione:

- <https://jdk.java.net/java-se-ri/15>

- **JSON:** acronimo di JavaScript Object Notation, è un formato di conservazione e invio di dati. Si basa su oggetti, ovvero coppie chiave/valore, e supporta i tipi booleano, stringa, numero, e lista. È semplice e leggibile ad occhio umano, inoltre non necessita di alcun processo di compilazione particolare per essere modificato. Nel software viene utilizzato come persistenza per la conservazione di dati.

- Documentazione: <https://www.json.org/json-it.html>

- **Docker:** piattaforma software che permette di creare, testare e distribuire applicazioni. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Le componenti client e server del software sono distribuite in container istanziabili negli ambienti in cui si vuole eseguire il programma.

- Versione utilizzata: 19.03.*
- Documentazione: <https://docs.docker.com/>

Per il download e l'installazione si rimanda al seguente link: <https://docs.docker.com/get-docker/>

- **Installazione con Linux**

Per poter installare Docker su Linux (avendo snap installato) è sufficiente eseguire: `sudo snap install docker`.

- **Installazione con Mac**

Per poter installare Docker su Mac è necessario scaricare il file installante, prestando attenzione alla versione del processore della quale si dispone, a questo link: <https://docs.docker.com/docker-for-mac/install/>.

Una volta scaricato basta avviarlo e procedere con l'installazione.



Per ulteriori informazioni leggere qui: <https://docs.docker.com/docker-for-mac/install/#install-and-run-docker-desktop-on-mac>.

– **Installazione con Windows**

Per poter installare Docker su Windows è necessario scaricare il file installante (direttamente da qui: <https://desktop.docker.com/win/stable/amd64/Docker%20Desktop%20Installer.exe>) e avviarlo, dopo essersi assicurati che sia abilitata l'opzione per la funzionalità di Windows Hyper-V o sarà necessario installare i componenti Windows necessari per WSL 2 e che siano selezionati nella pagina di "Configurazione". Per ulteriori informazioni leggere qui: <https://docs.docker.com/docker-for-windows/install/#install-docker-desktop-on-windows>.

- **Gradle**: sistema open source per l'automazione dello sviluppo fondato sulle idee di Apache Ant e Apache Maven, introduce un domain-specific language (DSL) basato su Groovy, al posto della modalità XML usata da Apache Maven per dichiarare la configurazione del progetto. Proprio come avviene con Apache Maven, la struttura di Gradle è costituita da un nucleo astratto e da una serie di plugin che ne espandono le funzionalità; al contrario di Maven, però, offre possibilità di definire il meccanismo di costruzione in linguaggio Groovy, nel file build, file che risulterà più leggero dell'equivalente XML e con una notazione più compatta per descrivere le dipendenze.
 - Versione utilizzata: 6.7
 - Documentazione: <https://docs.gradle.org/current/userguide/userguide.html>

Per il download e l'installazione si rimanda alla documentazione:

- <https://docs.gradle.org/current/userguide/installation.html>

2.1.2 Librerie e Framework

- **Spring**: framework_G open source per lo sviluppo di applicazioni su piattaforma Java. Fornisce una serie completa di strumenti per gestire la complessità dello sviluppo software, fornendo un approccio semplificato ai più comuni problemi di sviluppo e di testing; inoltre, grazie alla sua struttura estremamente modulare, è possibile utilizzarlo nella sua interezza o solo in parte, senza stravolgere l'architettura del progetto.
 - Documentazione: <https://spring.io/>
- **Gson**: libreria Java che può essere utilizzata per convertire gli oggetti Java nella loro rappresentazione JSON. Può anche essere utilizzato per convertire una stringa JSON in un oggetto Java equivalente. Nel contesto del software, viene utilizzato per la serializzazione e deserializzazione dei file Json della persistenza.
 - Documentazione: <https://github.com/google/gson/blob/master/UserGuide.md>
- **JUnit**: framework_G di unit testing per il linguaggio di programmazione Java.
 - Versione utilizzata: 5.7
 - Documentazione: <https://junit.org/junit5/docs/current/user-guide/>
- **Mockito**: framework_G open source di test per il linguaggio di programmazione Java. Il framework_G consente la creazione di oggetti fittizi (Mock Objects) in test di unità automatizzati. I Mock Objects sono oggetti simulati che imitano il comportamento di componenti reali in un ambiente controllato.



- Versione utilizzata: 3.*
- Documentazione: <https://junit.org/junit5/docs/current/user-guide/>

2.2 Client

2.2.1 Tecnologie

- **Node.js:** runtime system open source multiplatforma orientato agli eventi per l'esecuzione di codice Javascript. Molti dei suoi moduli base sono scritti in Javascript.
 - Versione utilizzata: 14.15.5
 - Link per download: <https://nodejs.org/it/download/>
- **HTML:** linguaggio markup per la strutturazione di pagine web. Viene utilizzato insieme ad Angular per la costruzione della struttura della web app.
- **CSS:** linguaggio utilizzato per definire la formattazione di documenti HTML, XHTML e XML, ad esempio i siti web e le relative pagine web. L'uso del CSS permette la separazione dei contenuti delle pagine HTML dal loro layout e permette una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione. Viene utilizzato insieme ad Angular per la stilizzazione degli elementi HTML.
- **Typescript:** linguaggio di programmazione open-source. Si tratta di un super-set di JavaScript che basa le sue caratteristiche su ECMAScript 6. Il linguaggio estende la sintassi di JavaScript in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica. È progettato per lo sviluppo di grandi applicazioni ed è destinato a essere compilato in JavaScript per poter essere interpretato da qualunque web browser o app. Viene utilizzato insieme ad Angular per la codifica del comportamento della webapp.
 - Versione utilizzata: 3.8 o superiore.

2.2.2 Librerie e Framework

- **Angular:** framework open source per lo sviluppo di applicazioni web a single page. Esso si basa sul pattern MVVM. Si basa sul linguaggio di programmazione TypeScript e sulla creazione di componenti che costruiscono la pagina web. Le applicazioni sviluppate in Angular vengono eseguite interamente dal web browser dopo essere state scaricate dal web server (elaborazione lato client). Questo comporta il risparmio di dover spedire indietro la pagina web al web-server ogni volta che c'è una richiesta di un'azione da parte dell'utente.
 - Versione utilizzata: 11.2.0
 - Link per installazione: <https://angular.io/guide/setup-local>
- **PrimeNG:** libreria per Angular per personalizzare i componenti così da creare un'interfaccia utente più accattivante.
 - Link per installazione: <https://primefaces.org/primeng/showcase/#/setup>



2.3 Version Control System e Continuous Integration

2.3.1 Git e gitflow

Git è un sistema di controllo per il versionamento veloce ed efficiente_G. Gitflow è un workflow che aiuta lo sviluppo software dando delle linee guida sui branch che strutturano le repo_A e le operazioni per l'implementazione di feature e rilascio di releases.

Maggiori informazioni:

- **Git:** <https://git-scm.com/>;
- **gitflow:** <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.

2.3.2 GitHub

GitHub è un provider di hosting internet per lo sviluppo di software e il controllo della versione utilizzando Git. Fornisce un intero ecosistema di strumenti (version control, issue tracking, project boards, continuous integration_G e delivery...) e permette la creazione di account personali o di organizzazioni.

- **Maggiori informazioni su GitHub:** <https://github.com/about>;
- **Three Way Milkshake su GitHub:** <https://github.com/Three-Way-Milkshake>.

2.3.3 GitHub Actions

È uno strumento integrato in ogni repo_A di GitHub, permette di creare singole attività_G combinabili al fine di realizzare complessi workflow personalizzati. Si possono creare nuove *actions* ed eventualmente pubblicarle, o utilizzare il vasto catalogo di automazioni già realizzate dalla community.

- **Documentazione:** <https://docs.github.com/en/actions>.



3 Setup

PORTACS viene distribuito tramite container Docker, per cui i dispositivi sui quali dovrà eseguire avranno minimi requisiti software. È possibile utilizzare i container anche per la fase_G di sviluppo, altrimenti si possono scaricare ed utilizzare gli strumenti descritti in [2.1.1](#) per l'esecuzione diretta in locale. PORTACS_A si divide su tre immagini Docker:

1. server;
2. client muletto (forklift);
3. client utente (user).

3.1 Requisiti di sistema

Sotto elencati saranno descritti i requisiti minimi del sistema per un corretto funzionamento del software PORTACS_A.

3.1.1 Requisiti Hardware

- Client → unità:
 - CPU_A dual-core o maggiore con frequenza indicativa ≥ 3 Ghz;
 - memoria RAM_A ≥ 4 GB dedicati;
 - connessione con bassi tempi di risposta.
- Client → admin o responsabile:
 - CPU_A dual-core o maggiore con frequenza indicativa ≥ 3 Ghz;
 - memoria RAM_A ≥ 4 GB dedicati;
 - connessione con bassi tempi di risposta.
- Server:
 - CPU_A quad-core o maggiore con frequenza indicativa ≥ 4 Ghz;
 - memoria RAM_A ≥ 8 GB dedicati;
 - connessione con bassi tempi di risposta.

3.1.2 Requisiti Software

3.1.2.1 Esecuzione

- Utilizzando Docker, il sistema operativo host non dovrebbe essere un problema, tuttavia in seguito si daranno le istruzioni per l'esecuzione valide in ambiente unix, nel quale si è anche effettuato la maggior parte dei test (Linux kernel v5.8 o MacOS v10.15);
- Docker (v19.03.*), installazione base, per esempio in Linux è sufficiente: `sudo snap install docker`;
- Google Chrome (v90).



3.1.2.2 Sviluppo

Vedi § [2.1.1](#)

3.2 Installazione ed avvio degli applicativi

3.2.1 Server

Nella macchina da utilizzare come server andrà scaricata l'immagine di portacs-server con la versione desiderata dal [docker hub di Three Way Milkshake](#). Per l'avvio, in ambiente Linux e MacOS:

- predisporre una cartella nella quale posizionarsi per l'esecuzione che contenga una cartella database che manterrà la persistenza, questa dovrà contenere i file map.json, forklifts.json e users.json;
- posizionarsi su tale cartella;
- eseguire:

```
docker run --rm --name server \  
-v $(pwd)/database:/database \  
-p 1723:1723 \  
threewaymilkshake/portacs-server
```

Il container verrà avviato nella finestra corrente e si potranno osservare i log del server, questo può essere terminato dalla stessa finestra con la combinazione Ctrl+C. Se si vuole eseguire il container in background è sufficiente aggiungere l'opzione -d, per terminarlo a questo punto bisognerà eseguire: docker stop server. Si raccomanda di impostare un ip statico per la macchina server, così da non dover richiedere variazioni frequenti nelle configurazioni dei client.

3.2.2 Client

Come per il server, scaricare l'immagine dal [docker hub di Three Way Milkshake](#) relativa al client voluto (forklift o user). Dopodiché posizionarsi in una cartella dove predisporre un file di configurazione config.txt:

- per i client *user* sarà sufficiente specificare l'indirizzo IP della macchina server così:
SERVER_ADDR=ip;
- per i client forklift, oltre all'indirizzo del server come per gli utenti, bisognerà aggiungere altre informazioni:
 - **obbligatorie:**
 - * id e token del muletto;
 - * posizione iniziale (corrispondente alla base dove il muletto verrà posizionato al primo avvio);
 - **opzionali:**
 - * per specificare delle porte diverse da far utilizzare ai servizi, quest'operazione è necessaria solo se si vogliono simulare più unità nella stessa macchina locale, al fine di evitare collisioni tra le porte;



- * bisogna impostare due campi, ngPort1 e ngPort2, devono essere diversi da altre configurazioni in uso sulla stessa macchina;
- * bisogna predisporre un file diverso per ogni unità che si vuole simulare.

Dev'essere poi sempre presente configurationEnv=custom. L'ordine in cui vengono specificati i campi non è importante. Segue un esempio di configurazione ben formata con utilizzo delle impostazioni opzionali:

```
startX=0
startY=1
ngPort1=4203
ngPort2=8083
forkliftId=f3
forkliftToken=abc
configurationEnv=custom
SERVER_ADDR=192.168.3.129
```

Queste informazioni sono a disposizione degli admin.

Per avviare un muletto sarà sufficiente eseguire (i valori di default per ngPort1 e ngPort2 sono rispettivamente 4201 e 8081):

```
docker run --rm --name <nome> -d \
  --env-file config.txt \
  -p 4201:4201 \
  -p 8081:8081 \
  portacs-client-forklift
```

Il <nome> specificato dopo --name servirà per terminare il container.

Per avviare un utente invece:

```
docker run --rm --name user -d \
  --env-file user.txt \
  -p 4300:4300 \
  -p 8090:8090 \
  portacs-client-user
```

3.2.2.1 Terminazione dei container

Se si vuole terminare un container l'istruzione è: `docker stop <nome>`, dove <nome> è il parametro che si ha specificato dopo --name. La specifica del nome all'avvio del container non è obbligatoria, nel caso di esecuzione con opzione -d verrà stampato l'id che si potrà usare al posto del nome per terminare il container o in ogni caso si possono ispezionare i container attivi tramite `docker ps`.



3.3 Costruzione degli artefatti e delle immagini

Quando il codice sorgente viene modificato le immagini docker corrispondenti devono essere ricostruite e prima di ciò bisogna produrre gli artefatti necessari. Per le operazioni descritte in seguito si assume di avere le cartelle con i sorgenti relativi.

3.3.1 Server

Per produrre l'unico artefatto `jar` necessario, si deve eseguire `./gradlew build`, questo produrrà appunto l'eseguibile segnandolo con la versione specificata in `build.gradle`. Questo comando esegue anche tutti i test di unità segnalando eventuali fallimenti. La build ha come precondizione `G` la corretta formattazione del codice, la quale può essere ottenuta con `./gradlew spotlessApply`. Con il `jar` aggiornato si può eseguire la build di docker per ottenere l'immagine aggiornata:

```
docker build -t threewaymilkshake/portacs-server .
```

3.3.2 Client

Per entrambi i client bisogna assicurarsi che il file `node_package.json` nelle cartelle corrispondenti contenga tutte e sole le dipendenze necessarie a node. Dopodiché bisogna compilare la parte angular:

- per i muletti: `ng build -c dev1`;
- per gli utenti: `ng build`.

A questo punto si possono generare le rispettive immagini:

```
docker build -t threewaymilkshake/portacs-client-forklift .  
docker build -t threewaymilkshake/portacs-client-user .
```

3.3.3 Caricamento nel docker hub

Per aggiornare le immagini nel docker hub, è necessario avere le credenziali o essere collaboratori. Dopo aver effettuato l'accesso tramite `docker login`, è sufficiente eseguire: `docker push <nome-immagine>`



4 Testing

Questo capitolo ha lo scopo di indicare agli sviluppatori come controllare in che modo opera il codice e la sua sintassi. Vengono di seguito esposti gli strumenti utilizzati per effettuare i test di analisi statica e dinamica del nostro applicativo.

4.1 Coveralls

Per quanto riguarda l'attività di code coverage_G è stato scelto di utilizzare *Coveralls*. *Coveralls* esegue revisioni automatiche al codice relative all'analisi statica attraverso *GitHub Actions*.

4.2 GitHub Actions

Il servizio di Continuous Integration che è stato deciso di utilizzare è *GitHub Actions*, fornito da *GitHub*.

GitHub Actions permette di creare dei workflow, ovvero processi automatici, con l'obiettivo di automatizzare il ciclo di vita dello sviluppo del software.

4.3 Testing lato server

4.3.1 Junit

Un framework_G di unit testing per la programmazione java. Junit viene utilizzato per i test di unità relativi al codice in Java. Per eseguire tutti i test è sufficiente lanciare il comando `./gradlew test`.

4.4 Testing lato client

4.4.1 Jasmine

Jasmine è un framework_G di unit testing per il codice JavaScript. Viene utilizzato per i test di unità relativi al codice in Angular CLI. Si scrivono i test, in un formato leggibile ad occhio umano, e si eseguono attraverso dei file formato js, collegati a relativi file html e css.

4.4.2 Karma

Karma è un test runner, che permette di utilizzare Jasmine in maniera più efficace_G. Karma e Jasmin vengono utilizzati in quanto framework_G di default per unit testing di Angular CLI.

4.4.3 Mocha

Mocha è un framework_G di unit testing per il codice JavaScript. Viene utilizzato per i test di unità relativi al codice in NodeJS.

5 Architettura del sistema

Il software è organizzato secondo un'architettura di tipo *client-server*. Nelle sezioni che seguono vengono presentate nel dettaglio le componenti client e server tramite diagrammi delle classi e descrizioni testuali sulla loro struttura e funzionamento.

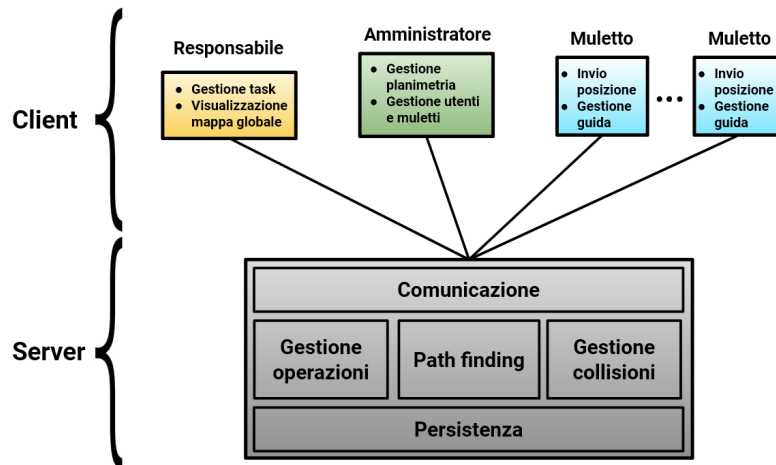


Figura 5.0.1: Rappresentazione ad alto livello dell'architettura del software



5.1 Server

L'architettura della componente server si articola in una 3-layer architecture, in cui si identificano i seguenti layer:

- communication layer;
- business layer;
- persistence layer.

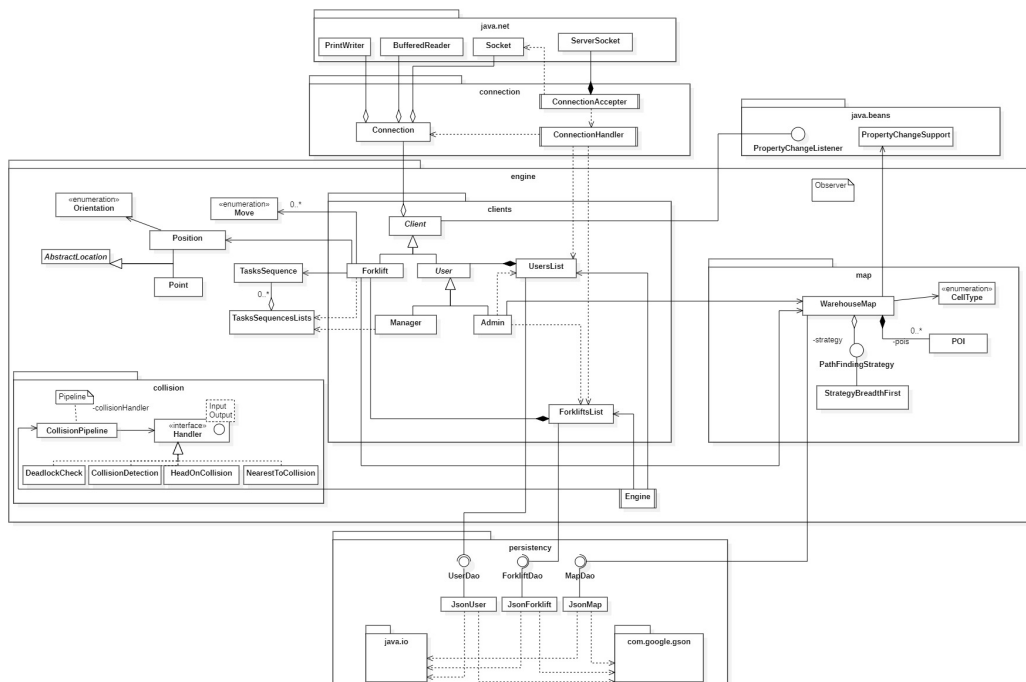


Figura 5.1.1: Visione complessiva dell'architettura del server

Le sezioni che seguono illustrano la struttura di ogni layer.

5.1.1 Communication layer

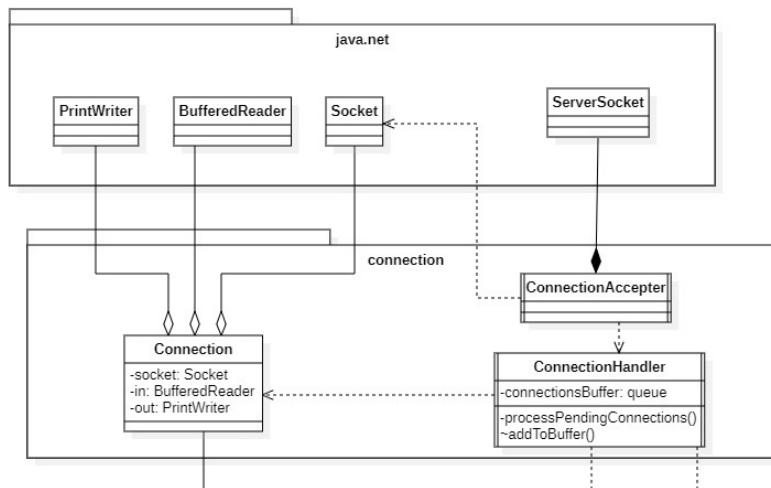


Figura 5.1.2: Visione di dettaglio del Communication Layer

Questo layer si interfaccia con i client esterni e ha lo scopo di gestire la comunicazione con questi. In particolare, la classe **ConnectionAcceptor** si occupa di accettare le nuove connessioni entranti tramite **ServerSocket**: essa esegue su un thread dedicato in modo da non bloccare le altre operazioni all'arrivo di una nuova connessione.

Per ogni nuova connessione, crea un oggetto **Socket** che passa a **ConnectionHandler**. Quest'ultima è una componente che esegue su un altro thread dedicato: rimane in attesa fino al risveglio determinato da **ConnectionAcceptor**: una volta attivato, procede a svuotare il buffer di **Socket** per creare oggetti di tipo **Connection**, istanziando per ognuno i buffer di input e output. Segue quindi il processo di autenticazione dei muletti o degli utenti, al termine del quale **ConnectionHandler** torna in attesa.

5.1.1.1 Incrementare il livello di sicurezza

Essendo **PORTACS_A** inserito in un contesto limitato al campo di operazione e non pubblico, gode di una sicurezza intrinseca essendo la rete privata e non esposta o accedibile all'esterno. Tuttavia all'interno della rete, le comunicazioni tra server e clients avvengono in chiaro. Se si vuole più sicurezza ci si può dotare di un certificato **TLS_A** e sostituire l'implementazione di **ServerSocket** e **Socket** rispettivamente con **SSLServerSocket** e **SSLSocket**. Dopodiché si dovrà sostituire nella parte node dei client la libreria **net** con quella **tls** e configurare il certificato. Non saranno necessarie ulteriori variazioni. Maggiori informazioni:

- [SSLServerSocket](#);
- [SSLSocket](#);
- [blog oracle sui certificati](#);
- [libreria TLS_A per node](#).



5.1.2 Business layer

Nel Business layer risiede il nucleo di elaborazione dei dati ricevuti dal layer superiore: i domini principali di cui si occupa sono:

- gestione della mappa e path finding;
- autenticazione dei client;
- elaborazione delle richieste;
- gestione delle tasks e dei POI_A;
- rilevazione e risoluzione delle collisioni.

Per facilitare la consultazione, lo studio di questo layer si concentra separatamente sui package di cui si compone. Per una visione dall'alto, riferirsi al diagramma complessivo all'inizio della sezione 5.1.



5.1.2.1 Clients

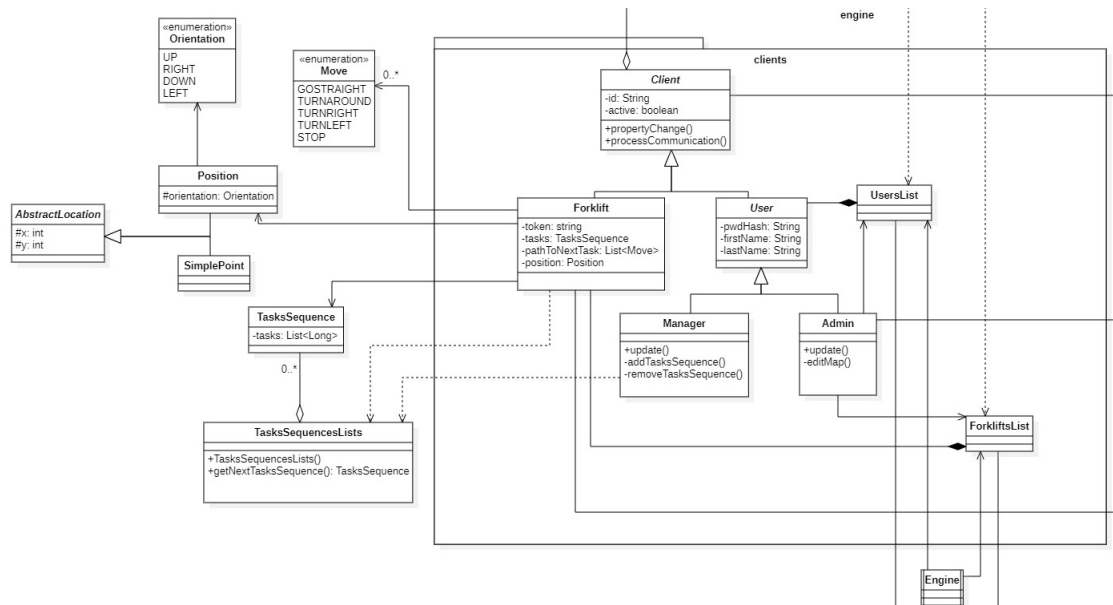


Figura 5.1.3: Visione di dettaglio del package Clients

La gerarchia dei Client prevede una prima suddivisione tra Forklift e User (muletto e utente), gli User si specializzano ulteriormente in Manager (responsabile) e Admin (amministratore). Viene mantenuto a runtime lo stato di tutti i muletti e degli utenti registrati rispettivamente in ForkliftsList e UsersList così da ridurre le operazioni sulla persistenza.

I Forklift si caratterizzano dagli attributi:

- position: rappresenta la posizione e orientamento attuali del muletto nella mappa;
- tasks: sequenza di task_G da compiere;
- pathToNextTask: una lista di mosse atte a raggiungere il prossimo POI_A (e quindi evadere la prossima task).

Notare che ogni Client possiede un attributo di tipo Connection, attraverso il quale viene regolata la comunicazione tramite Socket (per i dettagli si rimanda alle § 5.1.1 e 5.3). Questo viene assegnato all'autenticazione e può cambiare un numero indefinito di volte, ogni qual volta che la connessione con il client verrà chiusa per qualsiasi ragione alla riconnessione questa verrà correttamente abbinata alla sua istanza.

La classe Engine è il cuore del motore di calcolo: essa esegue su un thread dedicato e tramite un timer scandisce l'esecuzione temporizzata dell'elaborazione. In particolare, interroga periodicamente ForkliftsList e UsersList con i seguenti obiettivi:

- ricevere le nuove posizioni dai muletti;
- inviare le nuove informazioni agli utenti per la visualizzazione nel monitor real-time;



- rispondere ad eventuali altre richieste dell'iterazione precedente, avendo nel frattempo completato la loro elaborazione (calcolo percorso, aggiunta task_G, modifica mappa).

Dopodichè la ForkliftsList viene utilizzata dal modulo di rilevazione e gestione delle collisioni.

In questo layer si concentra l'utilizzo del framework_G Spring, utilizzato per gestire le dipendenze: alcune classi di utilizzo frequente e condiviso come UsersList, ForkliftsList e TasksSequencesLists (quest'ultima contenente tutte le liste di task_G inserite dal responsabile) vengono istanziate tramite *Dependency Injection* sfruttando il meccanismo dei *Bean* di Spring.

5.1.2.1.1 Introdurre nuove tipologie di utenti

Per introdurre nuove tipologie di utenti, bisogna estendere la classe astratta User e concretizzare in classi concrete, erediterà i campi principali, si può aggiungere altri metodi e attributi, ma è necessario fare l'override del metodo ProcessCommunication, che si occupa di gestire la comunicazione, in particolare ricevere i messaggi, eseguire quello che viene richiesto e inviare i dati necessari.

5.1.2.2 Mappa

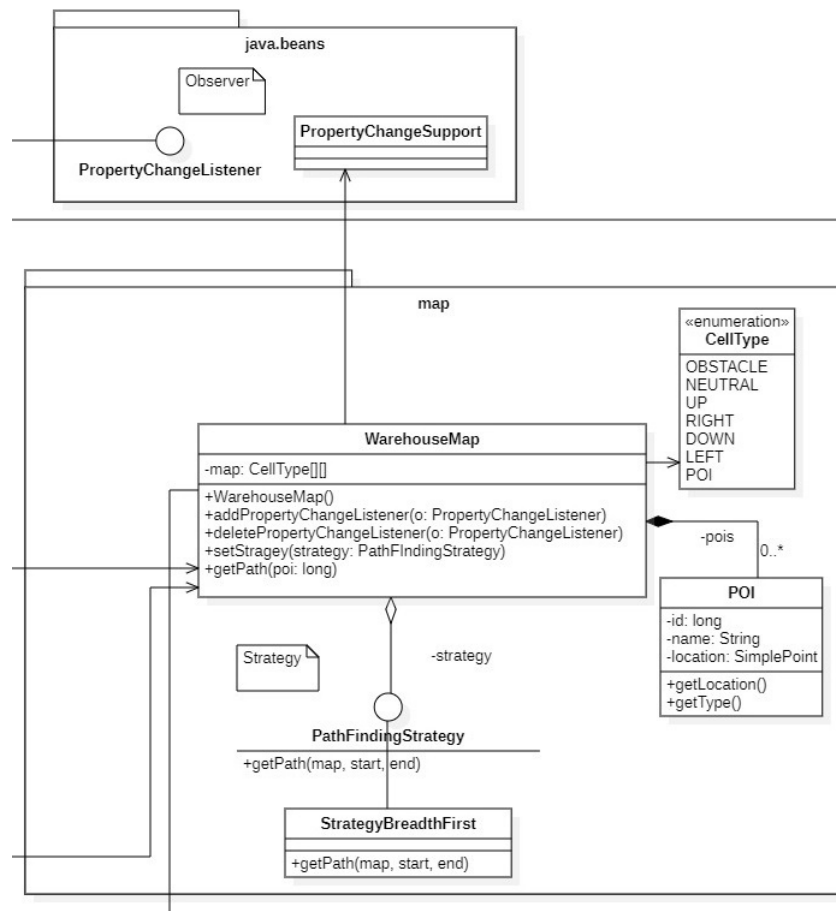


Figura 5.1.4: Visione di dettaglio del package Map

La classe WarehouseMap contiene la rappresentazione della planimetria_G del magazzino: essa è rappresentata tramite una matrice di CellType, campo di tipo enumerazione che esprime le caratteristiche di ogni frazione spaziale. Alla mappa è associata una lista di POI_A, e per ognuno la relativa locazione. Si osserva l'applicazione di alcuni design pattern_G:

- **observer**: tramite PropertyChangeSupport e PropertyChangeListener di java.beans viene applicato il pattern *observer*, definendo la WarehouseMap come Subject, e i Client come Observer: essi verranno notificati ad ogni cambiamento della stessa in modo che possano comunicarlo tramite le connessioni, così da riflettere le modifiche ed aggiornare le interfacce grafiche che visualizzano la mappa;
- **strategy**: per l'algoritmo di path finding attualmente viene implementata una strategia di tipo *breadth-first*, ma l'impostazione del pattern permette di aggiungere e variare



dinamicamente eventuali altre implementazioni aggiunte in futuro. WarehouseMap assume il ruolo di *context*, e i beneficiari sono i Forklift, i quali richiederanno il percorso ogni qualvolta si renderà necessario.

5.1.2.2.1 Estendere il path finding

Si può aggiungere un algoritmo alternativo di path finding, oltre a quello già implementato con una strategia breadth first (in ampiezza). Per farlo, bisogna creare una classe che implementi l'interfaccia PathfindingStrategy. Implementando questa interfaccia, viene creata una nuova strategia, e l'unica cosa necessaria da fare è implementare il metodo pubblico getPath, in quanto unico metodo utilizzato dall'esterno, che ritorna il percorso dal punto start a end sulla mappa, fatto di una list di moves (il nostro campo enum). La strategia può essere cambiata tramite il metodo setStrategy nella classe WarehouseMap e dunque cambiare quella di default nella configurazione di Spring, oppure si può aggiungere un'operazione per l'amministratore per cambiare quella di default a run-time.

5.1.2.3 Collisioni

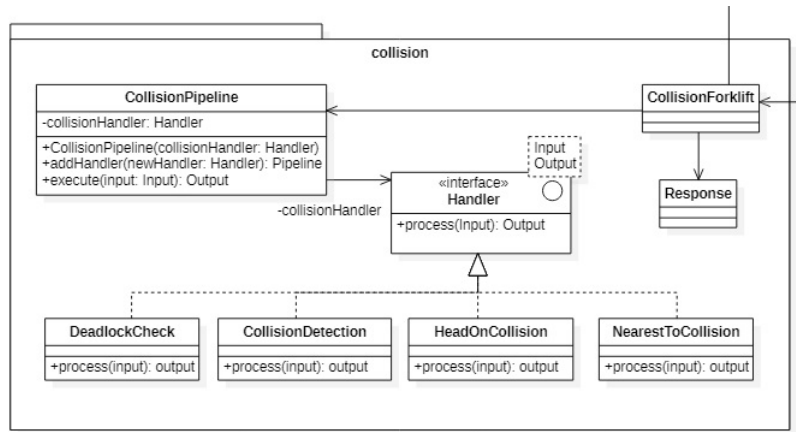


Figura 5.1.5: Visione di dettaglio del package Collision

Qui è contenuta la logica che gestisce le collisioni fra i muletti che circolano in guida autonoma all'interno del magazzino. L'elaborazione è scandita dal timer dell'Engine: ad ogni intervallo di tempo, vengono eseguite due operazioni sequenziali:

- **rilevazione degli stalli:** vengono individuate le unità che rimangono in stallo per un tempo superiore ad una soglia prestabilita, sintomo di una condizione critica nella circolazione dei muletti. La componente **DeadlockCheck** si occupa di gestire queste situazioni;
- **rilevazione delle collisioni:** conoscendo le future mosse di ogni unità, vengono determinate le possibili collisioni tra le stesse. La componente **CollisionDetection** esegue queste operazioni;
- **gestione collisioni frontali:** sulla base delle collisioni eventualmente rilevate, vengono gestite le potenziali collisioni frontali, che richiedono una risoluzione particolare. Questo step è svolto dalla componente **HeadOnCollisions**;
- **gestione delle precedenze:** l'ultimo passo risponde all'esigenza di stabilire delle precedenze durante la normale circolazione dei muletti: sulla base delle potenziali collisioni rilevate, alcuni si fermeranno per lasciare passare gli altri. Di ciò si occupa la componente **NearestToCollision**.

Viene applicato in questo contesto il design pattern `Chain Of Responsibility`¹, che permette di definire vari **Handler** da comporre come catena di operazioni. La pipeline può essere poi eseguita (se necessario, come in questo caso, ripetutamente) con un comando che attiva i vari step sequenzialmente. Ogni **Handler** specifica i tipi del proprio parametro di input e di output: l'output di un **Handler** sarà l'input dell'**Handler** successivo. In questo caso l'input sarà una struttura rappresentante muletti attivi con le loro posizioni e prossime mosse, mentre l'output un'altra struttura che ad ogni punto con collisione associi i muletti coinvolti e le azioni da intraprendere così da poterle comunicare.

¹ Variante del pattern *Chain Of Responsibility*: <https://java-design-patterns.com/patterns/pipeline/>

5.1.2.3.1 Modificare handler per la gestione delle collisioni

La struttura fornita dal design pattern *Pipeline*, come anticipato nella §5.1.2.3, consente di concatenare operazioni da eseguirsi sequenzialmente e il cui output di ognuna costituisce l'input della successiva. Per aggiungere un'operazione è necessario implementare l'interfaccia `Handler<I,O>` specificando i parametri di input (I) e di output (O) del nuovo `ConcreteHandler`. La logica dell'operazione si costruisce eseguendo l'*Override* del metodo `Process(I input) : O`. La costruzione della pipeline prevederà l'aggiunta del nuovo `ConcreteHandler` tramite il metodo `addHandler` in modo che l'esecuzione, avviata invocando il metodo `execute`, includa la nuova operazione nella sua sequenza.

L'applicazione di questo design pattern consente di modificare facilmente le operazioni che compongono la sequenza: se necessario, è possibile sostituirle anche tutte, cambiando di fatto l'implementazione dell'algoritmo; mantenendo però intatti i parametri di ingresso e uscita.

5.1.3 Persistence layer

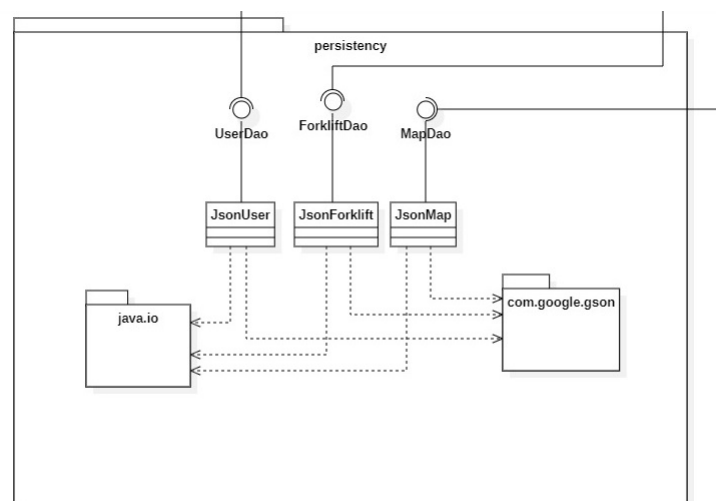


Figura 5.1.6: Visione di dettaglio del Persistence Layer

L'accesso a questo layer è regolato da 3 interfacce che gestiscono la persistenza delle tre tipologie di dati che vengono salvati:

- i dati e le credenziali degli utenti;
- gli identificativi ed i token di autenticazione dei muletti;
- la rappresentazione della mappa.

Ogni interfaccia si rivolge alla relativa componente del layer superiore che conserva a runtime i dati impiegati nell'esecuzione. La presenza delle interfacce favorisce il disaccoppiamento tra i moduli e permette di estendere a tipi di persistenza alternativi. Attualmente è implementato il salvataggio dei dati su file di tipo `.json`, viene fatto uso della libreria standard `java.io` e `GSON` per gestire l'interazione con questo tipo di tecnologia.



5.1.3.1 Implementare tipi di persistenza alternativi

Le interfacce UserDao, ForklistDao e MapDao forniscono i contratti sulla persistenza di utenti, muletti e mappa. Per implementare un nuovo tipo di persistenza bisogna creare una classe, implementando una di questa 3 interfacce, in base alla tipologia, e implementare i metodi richiesti dall'interfaccia. Similmente all'algoritmo di pathfinding, per settare questo tipo di persistenza bisogna passare per la configurazione Spring, per la gestione delle *dependency injection*.

5.2 Client

5.2.1 Muletti

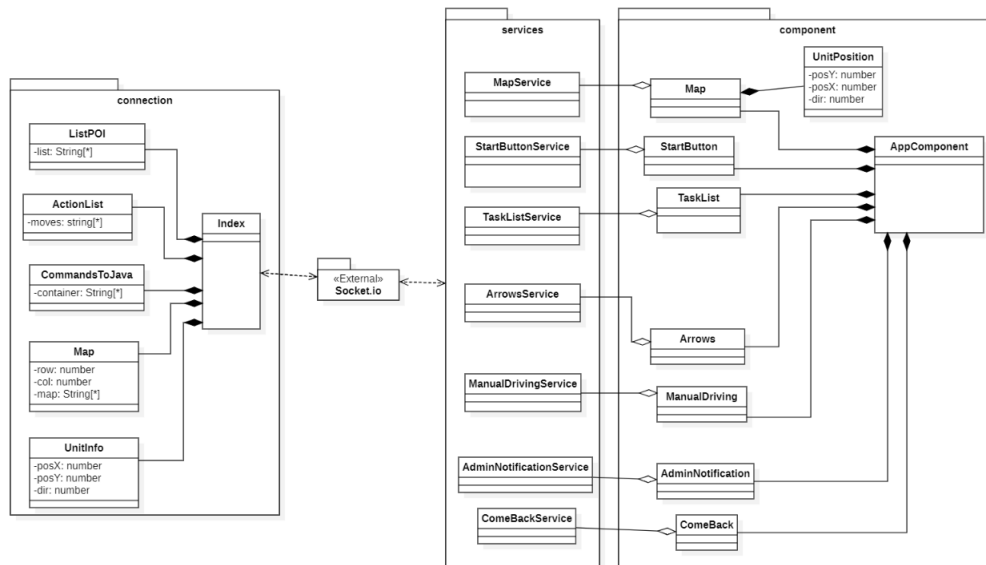


Figura 5.2.1: Diagramma UML_A delle classi per l'unità

Qui utilizziamo due tecnologie: Node per quanto riguarda il package connection e Angular per il resto. Queste due parti del front end comunicano attraverso il package esterno Socket.io, necessario gestire un flusso di dati attraverso i socket.

Il package services, come descritto anche nella documentazione di Angular, fa da intermediario tra il package connection e il package component, utilizzando degli Observer in ascolto di uno specifico socket e instradando l'informazione verso l'opportuno component.

Il package component permette di visualizzare sullo schermo le informazioni richieste grazie anche ai template di Angular. Ogni classe di questo package serve ad una specifica funzionalità:

- Map → visualizza la mappa del magazzino con la posizione in real time dell'unità;
- StartButton → mostra un bottone che serve a far partire l'unità;
- TaskList → mostra la lista di task_G che l'operatore dovrà compiere;
- Arrows → visualizza le azioni che compie l'unità in real time;
- ManualDrive → permette di cambiare guida da manuale ad automatica e viceversa, facendo visualizzare anche i pulsanti da premere per far muovere l'unità manualmente in caso;
- AdminNotification → visualizza un pulsante che, se premuto, notifica all'admin un evento eccezionale;
- ComeBack → mostra un pulsante alla fine del turno dell'operatore che, se premuto, guida automaticamente il muletto verso la propria base.



Il package connection, attraverso le classi Index e CommandsToJava, instaura inoltre una comunicazione TCP Socket con java, permettendo di creare una connessione tra frontend e backend.

5.2.2 Utenti (amministratori/responsabili)

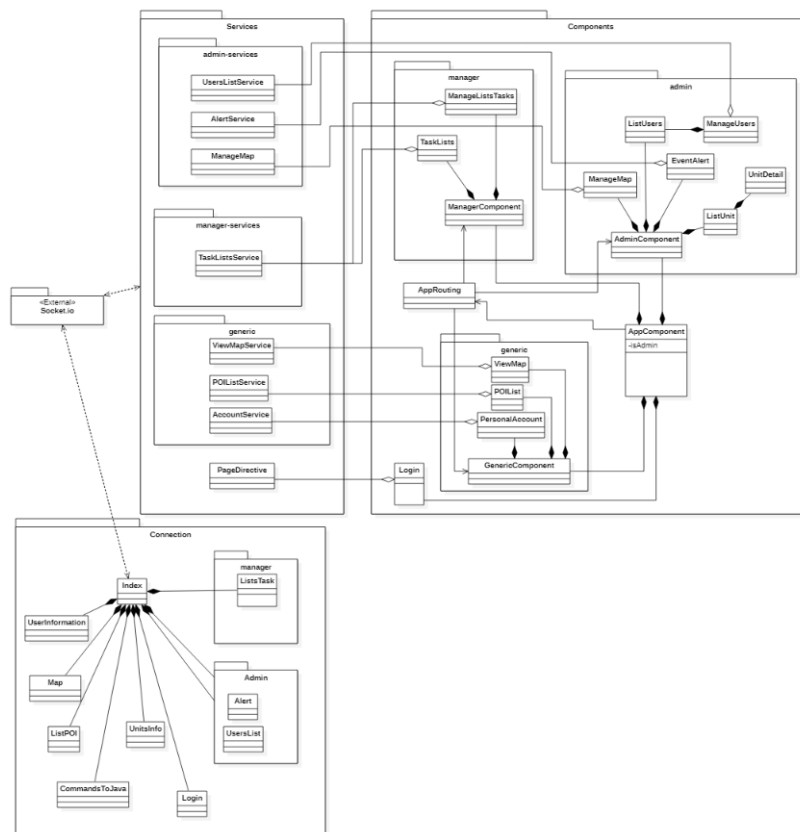


Figura 5.2.2: Diagramma UML_A delle classi per gli admin e i manager

Il diagramma di classe dell'admin-responsabile è molto simile a quello dell'unità: si avvale delle tecnologie Node, con il package connection, ed Angular, con tutti gli altri package.

Il contesto dei package è lo stesso dell'unità, di cui però si fa una differenziazione tra le funzionalità dell'admin e quelle del manager grazie alla classe Login e al routing di AppRouting presente nel package component: permette all'utente di effettuare il login, "attivando" solamente le funzionalità riferite al tipo di utente loggato.

Il package generic contiene classi di funzionalità condivise tra manager e admin.

Le classi presenti in component permettono di attivare certe classi riferite al package (e quindi alle funzionalità di uno specifico tipo di utente):

- admin:
 - ListUsers → aggiunta o rimozione di manager;



- EventAlert → visualizzazione eventi eccezionali;
 - ManageMap → modifica la planimetria_G e le caratteristiche della mappa;
- manager:
 - TaskLists → visualizzazione delle liste di task_G;
 - MangeListsTask → aggiunta, modifica e rimozione di liste di task_G;
- generic (sia per admin che per manager):
 - ViewMap e POIList → visualizzazione in real time di tutte le unità nel magazzino.

5.2.2.1 Introdurre nuovi tipi di utenti

Per introdurre nuovi tipi di utente basterà creare un nuovo package dentro Connection lato Node del sistema, inserendo tutte le funzionalità necessarie al tipo d'utente.

Successivamente bisognerà implementare le corrispondenti parti nel package Services e Component di Angular per la visualizzazione.

Infine bisognerà inserire il nuovo tipo di utente dentro la classe di Login.

5.3 Comunicazione

Le comunicazioni tra client e server avvengono tramite stringhe inviate sui TCPsocket che li connettono, secondo il protocollo descritto alla § 5.3.2. La creazione di un socket che viene mantenuto fino alla disconnessione permette di richiedere l'autenticazione di ogni client solo alla connessione, senza bisogno di scambi ulteriori di token o codici di sessioni, in quanto ogni client ha i suoi canali dedicati di input e output per cui si sa sempre a chi si scrive e da chi si legge. Le connessioni all'interno del server sono rappresentate con una classe dedicata, i client poi aggregano un attributo di tale classe: ciò permette di mantenere lo stato di questi anche in caso di disconnessione in quanto verrà distrutto solo l'oggetto connessione, e alla successiva autenticazione del client verrà correttamente abbinato lo stato interno che non avrà subito modifiche se non per l'associazione della nuova connessione.

5.3.1 Diagrammi di sequenza

Di seguito vengono riportati i diagrammi di sequenza.

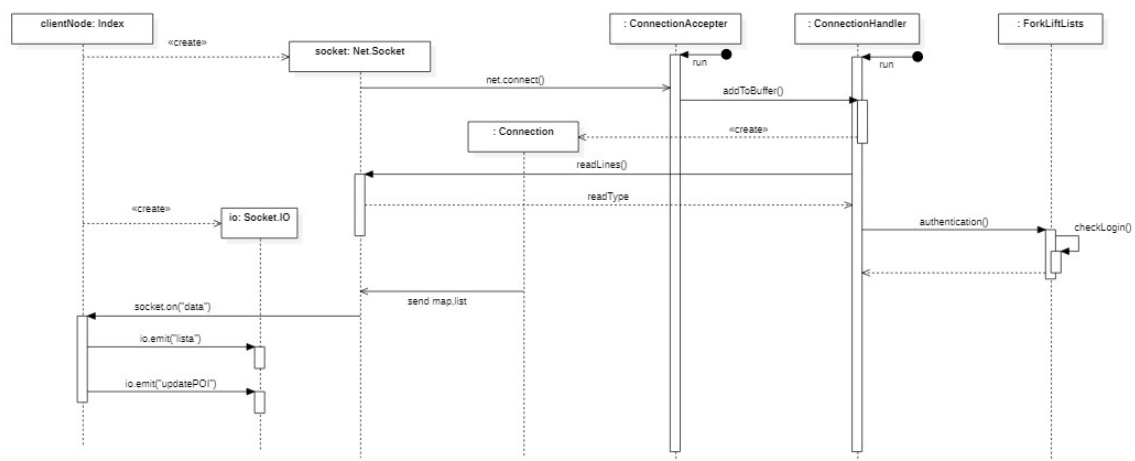


Figura 5.3.1: Visione complessiva della connessione server Java - unità muletto in NodeJS

Questo diagramma di sequenza rappresenta la prima connessione tra un'unità muletto in NodeJS e il server Java, attraverso il TCPsocket. Nel diagramma viene raffigurato anche un socket.IO, che mette in comunicazione NodeJS con l'interfaccia grafica realizzata in Angular (non rappresentata nel diagramma).

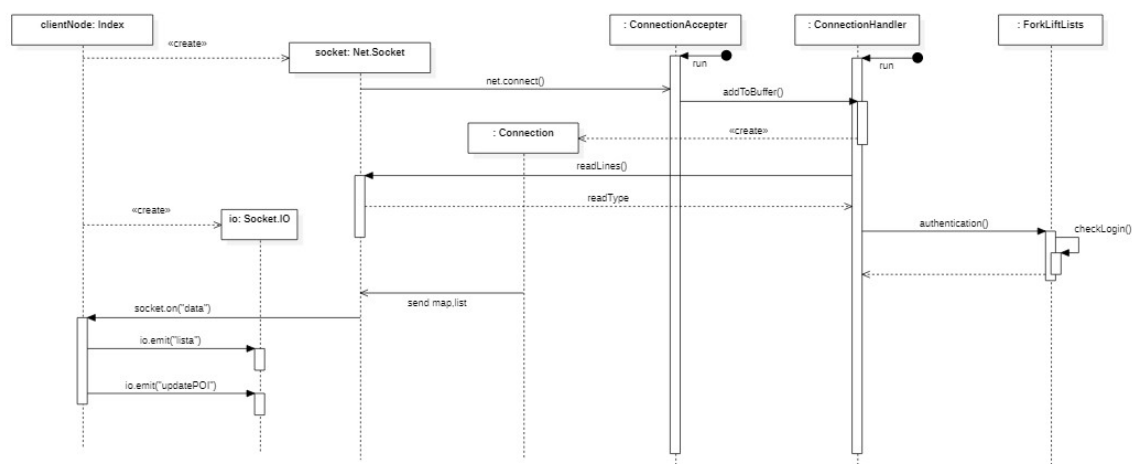


Figura 5.3.2: Esempio di comunicazione tra server e client, attraverso i socket TCP

Questo diagramma rappresenta un esempio di comunicazione tra server e client di tipo mulletto. Il server legge i dati inviati dal mulletto e invia una risposta, il mulletto aggiorna l'interfaccia grafica in base alla risposta ricevuta dal server.



5.3.2 Protocollo di comunicazione

Ogni stringa può contenere uno o più comandi, separati da ';' e ogni comando può avere 0 o più parametri, separati da ','.

Esempio sequenza: POS,1,1,0;PATH,1

5.3.2.1 Modificare il protocollo

Per la modifica di parti esistenti o per l'aggiunta di nuove funzionalità riguardo la comunicazione è sufficiente andare a modificare le corrispondenti sezioni. In base alla necessità:

Contesto	Lato Server	Lato Client
Connessione	ConnectionAcceptor.java: run	index.js: createConnectionServer
Autenticazione	ConnectionHandler.java: execute, auth di Users/ForkliftsList e conseguentemente authenticate di Forklift/User	index.js: createConnectionServer e quindi in net.connect
Modificare una funzionalità generica utente esistente	User.java: processCommuncation, all'interno dello switch, identificare la funzionalità voluta (esiste un case per ogni possibile messaggio ricevibile) ed apportare le modifiche desiderate	Trovare all'interno dello switch in index.js: createConnectionServer il case desiderato e modificarlo
Aggiungere una funzionalità generica utente	Aggiungere un case all'interno dello switch di cui sopra avendo cura di terminare con break; per evitare <i>fallthrough</i>	Aggiungere un case all'interno dello switch in index.js: createConnectionServer
Modificare una funzionalità amministratore / responsabile	Admin/Manager.java: processCommuncation, all'interno dello switch, identificare la funzionalità voluta (esiste un case per ogni possibile messaggio ricevibile) ed apportare le modifiche desiderate	Modificare il comportamento dell'opportuno case all'interno dello switch in index.js: createConnectionServer
Aggiungere una funzionalità amministratore / responsabile	Aggiungere un case all'interno dello switch di cui sopra avendo cura di terminare con break; per evitare <i>fallthrough</i>	Aggiungere un case all'interno dello switch in index.js: createConnectionServer
Modificare una funzionalità muletto	Forklift.java: processCommuncation, all'interno dello switch, identificare la funzionalità voluta (esiste un case per ogni possibile messaggio ricevibile) ed apportare le modifiche desiderate	Modificare il comportamento dell'opportuno case all'interno dello switch in index.js: createConnectionServer



Contesto	Lato Server	Lato Client
Aggiungere una funzionalità muletto	Aggiungere un case all'interno dello switch di cui sopra avendo cura di terminare con <code>break</code> ; per evitare <i>fallthrough</i>	Aggiungere un case all'interno dello switch in <code>index.js</code> : <code>createConnectionServer</code>

Tabella 5.3.1: Modificare il protocollo di comunicazione

5.3.2.1.1 Esempio modifica lato server

Volendo modificare il comportamento del comando RMU (remove user) sarà sufficiente andare nel metodo `processCommunication` all'interno di `Admin.java`, identificare il case "RMU" e al suo interno fare il *parsing* degli argomenti che ci si aspetta e processarli come voluto. Si consiglia di delegare queste operazioni ad altri metodi come visibile in gran parte del codice già esistente, per evitare di sovraccaricare `processCommunication` di responsabilità. Poi come si può osservare all'interno di `removeUser` quando si deve inviare una risposta al client (la quale dovrà essere ben formata, vedi inizio [5.3.2](#)) sarà sufficiente chiamare `connection.writeToBuffer(...)` passandogli il messaggio da inviare.

5.3.2.1.2 Esempio modifica lato client

Volendo modificare il comportamento del comando RMU (remove user) sarà sufficiente andare nel metodo `createCommunicationServer` all'interno di `index.js`, identificare il case "RMU" e al suo interno modificarne il comportamento, conoscendo il protocollo di comunicazione e come queste informazioni vengono passate.

5.3.2.2 Connessione: identificazione e login

Quando un client si connette deve essere identificato come tipo ed autenticato, perciò deve inviare separatamente ed in sequenza:

1. **TYPE:** FORKLIFT o USER;
2. **ID:** identificativo personale;
3. **PWD/TOKEN:** password o token a seconda che sia rispettivamente un utente o un muletto.

Quindi riceverà come risposta:

- nel caso di FORKLIFT: OK oppure FAIL,MSG
- nel caso di USER: OK,TYPE oppure FAIL,MSG
 - dove TYPE indica il ruolo dell'utente (ADMIN o MANAGER)

dove MSG conterrà maggiori dettagli sulla causa.

5.3.2.2.1 Esempio connessione ed autenticazione muletto

Dato un muletto con `id=f1` e `token=abcdef`:

- invia: `FORKLIFT\nf1\nabcdef`;
- riceve: OK oppure FAIL,messaggioErrore

Funzionamento analogo per gli utenti con password al posto di token.



5.3.2.3 Enumerazioni

In seguito si farà riferimento più volte ai diversi tipi enum presenti nella logica di business, per cui segue un riassunto:

ENUM				
↓Val \ Enum→	PoiType	Move	Orientation	CellType
0	LOAD	GOSTRAIGHT	UP	OBSTACLE
1	UNLOAD	TURNAROUND	RIGHT	NEUTRAL
2	EXIT	TURNRIGHT	DOWN	UP
3	–	TURNLEFT	LEFT	RIGHT
4	–	STOP	–	DOWN
5	–	–	–	LEFT
6	–	–	–	POI _A

Tabella 5.3.2: Riepilogo enumerazioni

5.3.2.4 Comandi clients → server

Per i comandi di risposta, controllare i corrispondenti in § 5.3.2.5

FORKLIFTS → SERVER		
Comando	Descrizione	Risposta
POS,X,Y,DIR	Posizione attuale del muletto, considerando la mappa come una matrice: <ul style="list-style-type: none"> • X: riga della matrice • Y: colonna ““ • DIR: orientamento assoluto secondo enum Orientation 	–
LIST	Richiede nuova lista di task _G da completare	LIST,...
PATH,C	Richiede il percorso migliore per raggiungere il POI _A della task _G corrente a partire dalla posizione attuale. Se C=1 rimuove la task _G e passa alla successiva	PATH,...
BASE	Segnala l'arrivo ad una base	–
ECC	Segnala il verificarsi di un evento eccezionale	–

Tabella 5.3.3: Comandi clients → server | Forklifts

USER generico → SERVER		
Comando	Descrizione	Risposta
EDIT,T,PAR	Modifica dati del proprio profilo <ul style="list-style-type: none"> • T: NAME/LAST/PWD • PAR: nuova valore per T 	–
LOGOUT	Richiesta di disconnessione	–

Tabella 5.3.4: Comandi clients → server | User generico

MANAGER → SERVER		
Comando	Descrizione	Risposta
ADL,P1,P2,...	Aggiunge una nuova lista di task _G <ul style="list-style-type: none"> P1...: id dei poi che compongono la lista 	ADL,...
RML, ID	Richiede la cancellazione della lista con id=ID	RML,...

Tabella 5.3.5: Comandi clients → server | User Manager (Responsabile)

ADMIN → SERVER		
Comando	Descrizione	Risposta
MAP,R,C,SEQ	Nuova planimetria _G <ul style="list-style-type: none"> R: num righe C: “ colonne SEQ: sequenza di interi corrispondenti all’enum CellType rappresentanti stati di una cella, indicanti la nuova planimetria_G, elencati per righe 	MAP,...
CELL,X,Y,A[,ID,T,NAME]	Modifica una cella, la parte tra [] è presente solo in caso di POI _A <ul style="list-style-type: none"> X e Y riga e colonna della matrice A: numero rappresentante l’azione da intraprendere, corentemente con CellType Solo nel caso di POI _A : <ul style="list-style-type: none"> ID: id del POI_A, 0 se in creazione; T: tipo di POI_A secondo PoiType NAME: stringa di caratteri da associare al POI_A 	CELL,...
ADU,T,NAME,LAST	aggiunge nuovo utente <ul style="list-style-type: none"> T: tipo (ADMIN o MANAGER) NAME e LAST: rispettivamente nome e cognome 	ADU,...

ADMIN → SERVER		
Comando	Descrizione	Risposta
RMU, ID	Rimuove l'utente con id=ID	RMU, ...
EDU, ID, A, PAR	Modifica l'utente con id=ID, con <ul style="list-style-type: none"> A: azione da intraprendere tra: <ul style="list-style-type: none"> – NAME: modifica nome – LAST: modifica cognome – RESET: esegue reset della password PAR: nuovo valore da assegnare (assente in caso di reset) 	EDU, ...
ADF, ID	Aggiunge nuovo muletto. ID: stringa che si vuole assegnare come identificativo al nuovo muletto (NON sarà più modificabile)	ADF, ...
RMF, ID	Rimuove il muletto con id=ID	RMF, ...
LISTF	Richiede lista di tutti i muletti registrati	LISTF, ...
LISTU	Richiede lista di tutti gli utenti registrati	LISTU, ...

Tabella 5.3.6: Comandi clients → server | User Admin (Amministratore)



5.3.2.5 Comandi server → clients

Tutti i comandi che possono ritornare un esito positivo o negativo hanno 2 varianti:

- CMD,OK,MORE: successo, eventualmente MORE contiene parametri di risposta aggiuntivi
- CMD,FAIL,MSG: fallimento, MSG contiene maggiori informazioni sulle cause.

SERVER → CLIENTS generici		
Comando	Descrizione	Risposta
MAP,R,C,SEQ	Indica la planimetria _G <ul style="list-style-type: none">• R e C: numero di righe e colonne• SEQ: sequenza di interi corrispondenti all'enum CellType rappresentanti stati di una cella, indicanti la nuova planimetria_G, elencati per righe	—
POI,N,X,Y,T,ID,NAME	Rappresenta tutti i poi, la parte da X in poi si ripete per ogni POI _A <ul style="list-style-type: none">• N: num totale POI_A• X,Y e T posizione nella matrice e tipo secondo PoiType• ID e NAME: rispettivamente identificativo e nome del POI_A	—

Tabella 5.3.7: Comandi server → clients | generici per tutti i client

SERVER → FORKLIFT		
Comando	Descrizione	Risposta
ALIVE	Ha lo scopo primario di verificare l'integrità della connessione ed ha come effetto l'ottenimento della nuova posizione. Se l'invio fallisce il muletto corrispondente viene considerato disconnesso, l'oggetto Connection relativo verrà chiuso e distrutto e il muletto dovrà riautenticarsi. Si presuppone che questo non si muova più finché la connessione non viene ristabilita, in quanto i suoi spostamenti sarebbero sconosciuti al server e questo non potrebbe intervenire per evitare eventuali collisioni. Ad esso possono seguire ulteriori comandi o risposte a comandi precedente, secondo la sintassi generale per cui separati da ','	POS,...
LIST, ID1, ID2...	Invia lista di task _G assegnate. ID1.. sono gli id dei POI _A da raggiungere	—
PATH, SEQ	Invia il percorso per raggiungere il prossimo POI _A , composto da mosse successive secondo Move, separate da ','	—
STOP, N	Richiede lo stop immediato del muletto per N istanti, che verranno contati alle ricezioni dei futuri ALIVE. Se N=0 stop indefinito fino alla ricezione di START.	—
START	Consente ad un muletto fermato a tempo indeterminato di ripartire	—

Tabella 5.3.8: Comandi server → clients | Forklifts (muletti)

SERVER → USER generico		
Comando	Descrizione	Risposta
UNI,N,ID1,X1,Y1,D1	<p>Indica le posizioni di tutti i muletti. Da ID1 in poi si ripete per ogni muletto</p> <ul style="list-style-type: none"> • N: num totale dei muletti attivi per i quali si sta ricevendo la posizione • IDn: id del muletto n • Xn, Yn e Dn: posizione rispetto alla matrice e orientamento secondo Orientation del muletto IDn. <p>Se l'invio di questo fallisce, l'utente viene considerato disconnesso.</p>	–
LIST,NF,IDF,N,IDP1,IDP2...	<p>Indica la lista di task_G presa in carico da un muletto</p> <ul style="list-style-type: none"> • NF: numero totale muletti dei quali si stanno ricevendo le task_G; • IDF: id del muletto a cui ci si sta riferendo • N: numero di task_G prese in carico • IDP1...: sequenza di id dei POI_A da raggiungere 	–

Tabella 5.3.9: Comandi server → clients | User generico

SERVER → MANAGER		
Comando	Descrizione	Risposta
ADL,OK,ID	Conferma aggiunta nuova lista di task _G . ID indica l'identificativo della nuova lista	–
ADL,FAIL,MSG	Segnala errore nella creazione di una nuova lista di task _G .	–
Funzionamento analogo per RML		

Tabella 5.3.10: Comandi server → clients | Utente Manager (responsabile)

SERVER → ADMIN		
Comando	Descrizione	Risposta
MAP,OK	Conferma successo modifica mappa	—
MAP,FAIL,MSG	Modifica mappa fallita	—
Analogamente a quelli sopra, stesso discorso vale per: CELL, RMU, RMF		
ADU,ID,PWD	In risposta alla creazione di un utente <ul style="list-style-type: none"> • ID che rappresenta il nuovo utente • PWD password temporanea per il nuovo utente, il quale è tenuto a cambiarla tempestivamente 	—
EDU,OK[,PWD]	Modifica utente avvenuta con successo. PWD contiene la nuova password in caso di richiesta di reset. Anche in questo caso, l'utente una volta ricevuta la password dall'admin è tenuto a reimpostarla.	—
EDU,FAIL,MSG	Modifica utente fallita	—
ADF,OK,TOKEN	Aggiunta di un nuovo muletto avvenuta con successo. Il TOKEN serve per la configurazione del nuovo muletto sul dispositivo client che verrà associato alla nuova unità	—
ADF,FAIL,MSG	Aggiunta nuovo muletto fallita (esiste già muletto con l'id richiesto)	—
LISTF,N,ID1,T1,ID2,T2...	In risposta alla richiesta della lista dei muletti: <ul style="list-style-type: none"> • N: num totale muletti • IDn, Tn: rispettivamente id e token del muletto n 	—
LISTU,N,ID1,UN1,UL1,R1...	In risposta alla richiesta della lista degli utenti: <ul style="list-style-type: none"> • N: num totale utenti • ID1,UN...: rispettivamente: identificativo, nome, cognome e ruolo 	—

Tabella 5.3.11: Comandi server → clients | Utente Admin (amministratore)