>Three Way Milkshake_

Manuale Manutentore

Three Way Milkshake - Progetto "PORTACS"

threewaymilkshake@gmail.com

Versione	1.0.0
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 appr.	Approvazione
1.0.0	Approvazione documento	2021-04-27	Greggio Nicolò

Vers.	Descrizione	Redazione	Data red.	Verifica	Data ver.
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

Elenco delle figure

Elenco delle tabelle

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

- Norme di progetto$_G$ v3.0.0 : per qualsiasi convenzione sulla nomenclatura degli elementi presenti all'interno del documento;
- Regolamento progetto$_G$ didattico:
  https://www.math.unipd.it/~tullio/IS-1/2020/Dispense/P1.pdf;
- 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 - 10^{th} 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$_G$.

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-14
  - Documentazione: https://docs.oracle.com/en/java/javase/14/

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

  - https://jdk.java.net/java-se-ri/14
- **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 containter instanziabili 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 alla documentazione:

  - https://docs.docker.com/get-docker/
- **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$_G$. 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$_G$.
  - 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 il la serializzazione e deserielizzazione 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 multipiattaforma 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$_G$ 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;
  - memoria Ram >= 4GB;
  - connessione con bassi tempi di risposta.
- Client → admin o responsabile:
  - CPU_A dual-core o maggiore;
  - memoria Ram >= 4GB;
  - connessione con bassi tempi di risposta.
- Server:
  - CPU_A quad-core o maggiore;
  - memoria Ram >= 8GB;
  - connessione con bassi tempi di risposta.

3.1.2 Requisiti Software

3.1.2.1 Esecuzione

- Docker (v19.03.*);
- 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 docker hub di Three Way Milkshake. Per l'avvio, in ambiente Linux e MacOS:

- predisporre una cartella che contenga un file `config.txt` ed una cartella `resources` che manterrà la persistenza;
- posizionarsi su tale cartella;
- eseguire:

```
docker run -v $(pwd)/resources:/resources \
    --env-file config.txt threewaymilkshake/portacs-server
```

All'interno del file `config.txt` si possono specificare (come coppie key=value) delle configurazioni diverse da quelle di default per quanto riguardo percorso della persistenza e porta sulla quale il server dovrà esporre il Server Socket per il collegamento. Se non si desiderano modificare i valori di default il file e la relativa parte nel comando possono essere omessi.

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 sul dispositivo che dovrà fungere da client andranno impostata la configurazione in un file `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 come per gli utenti, bisognerà aggiungere altre 2 righe per la configurazione di ogni muletto:
  - `ID=id del muletto`;
  - `TOKEN=token del muletto`.

  Queste informazioni sono a disposizione degli admin.

Dopodiché, in entrambi i casi, sarà sufficiente eseguire:

```
docker run --env-file config.txt threewaymilkshake/portacs-client-<type>
```

con `<type>` tra `forklift` o `user`.

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.

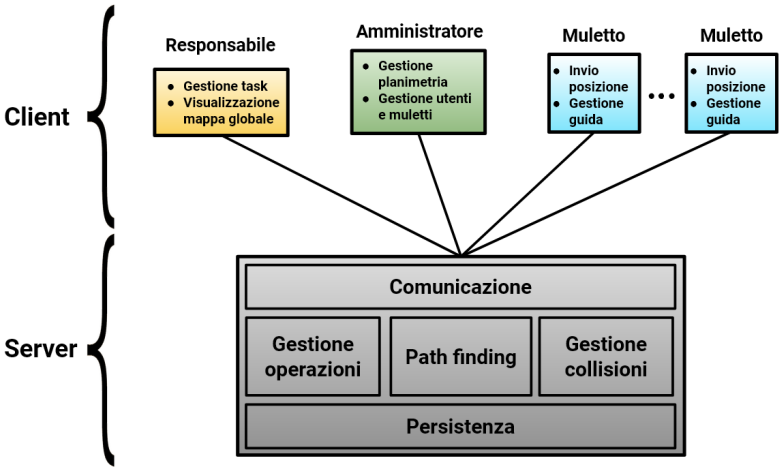

Figura 5.0.1: Rappresentazione ad alto livello dell'architettura del software

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.

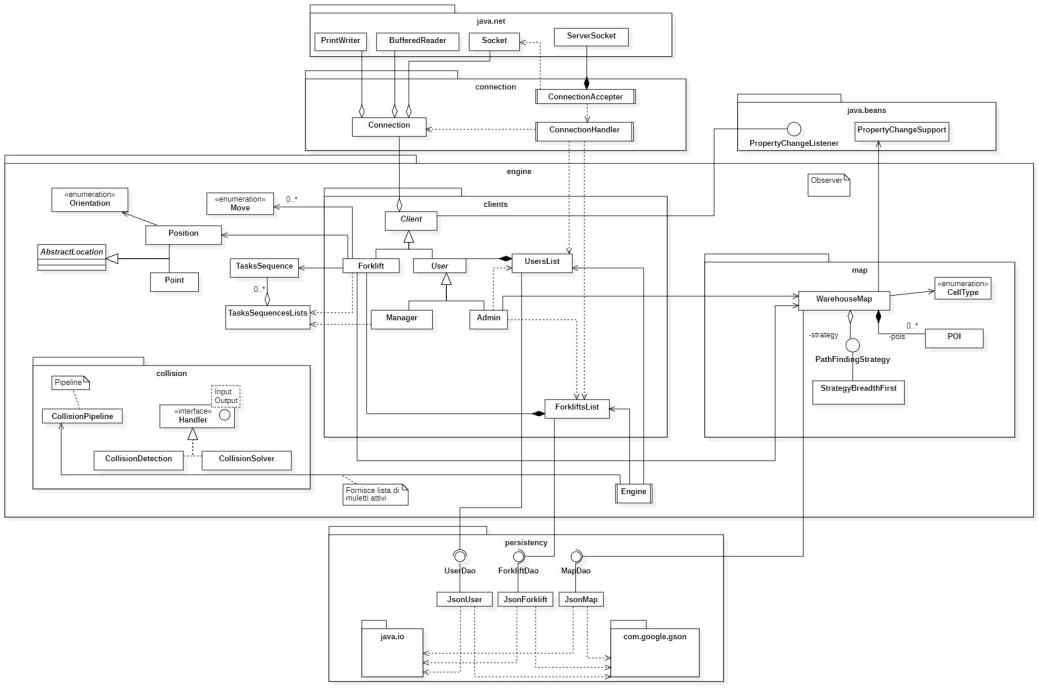

Figura 5.1.1: Visione complessiva dell'architettura del server

Le sezioni che seguono illustrano la struttura di ogni layer.

5.1.1 Communication layer

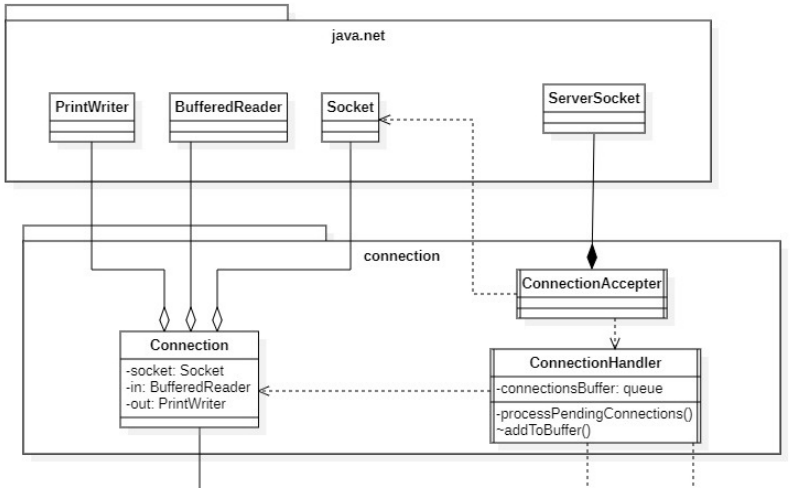

Figura 5.1.2: Visione di dettaglio del Communication Layer

Questo layer si interfaccia con i client esterni e ha lo scopo di gestire la comunicazione con questi. In particolare, la classe `ConnectionAccepter` 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 `ConnectionAccepter`: 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.

5.1.2 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

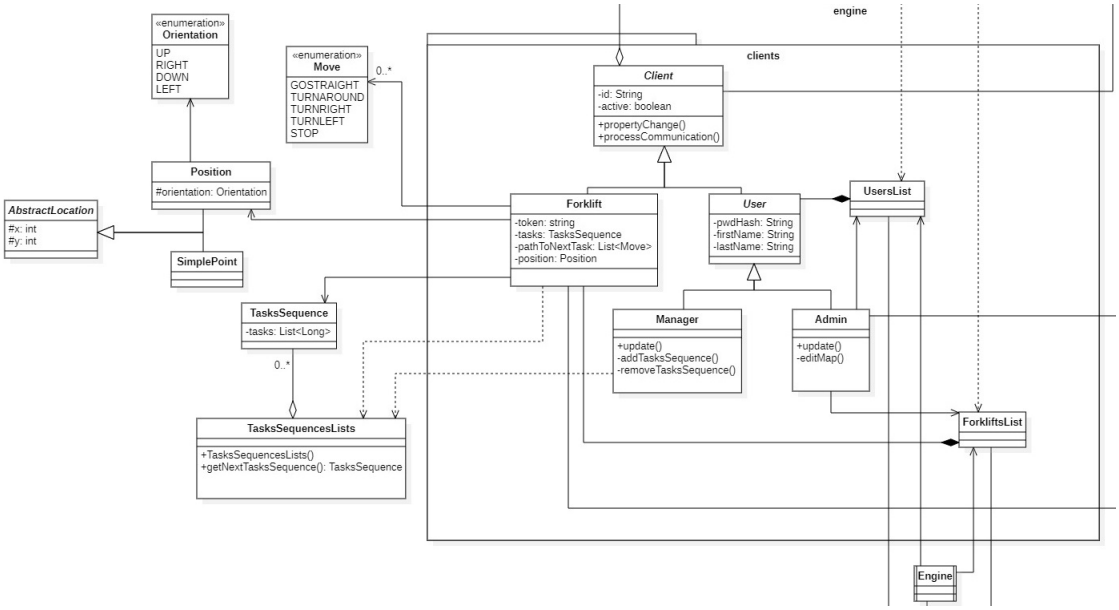

Figura 5.1.3: Visione di dettaglio del package Clients

La gerarchia dei `Client` prevede una prima suddivisione 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.

5.1.2.2 Mappa

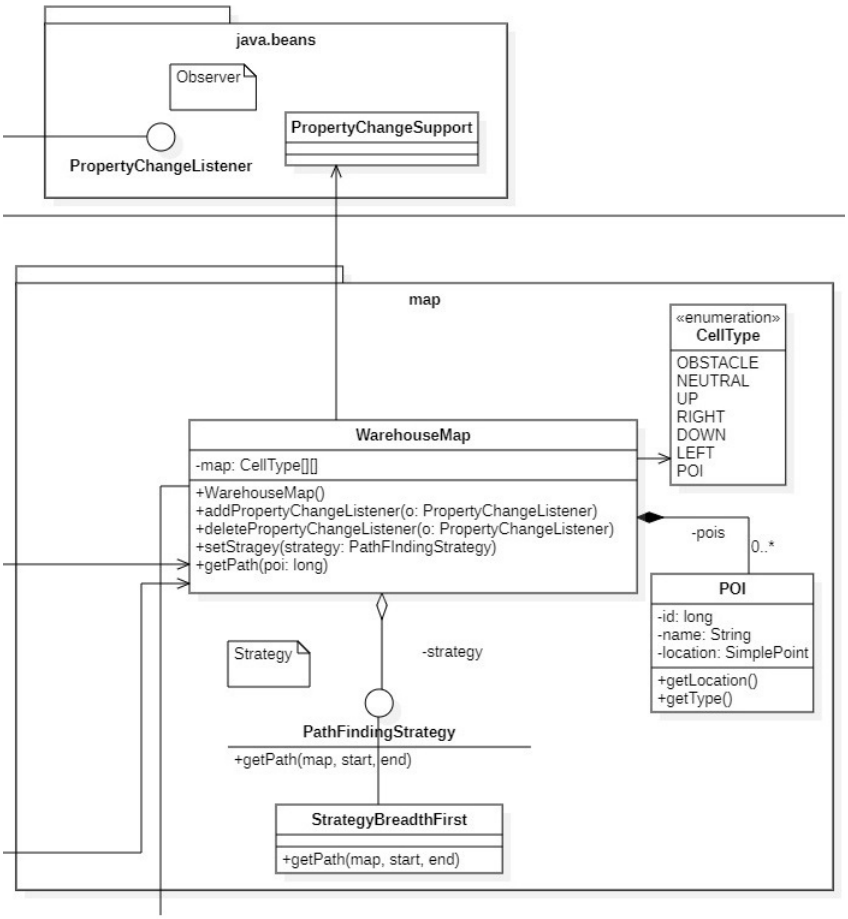

Figura 5.1.4: Visione di dettaglio del package Map

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.

5.1.2.3 Collisioni

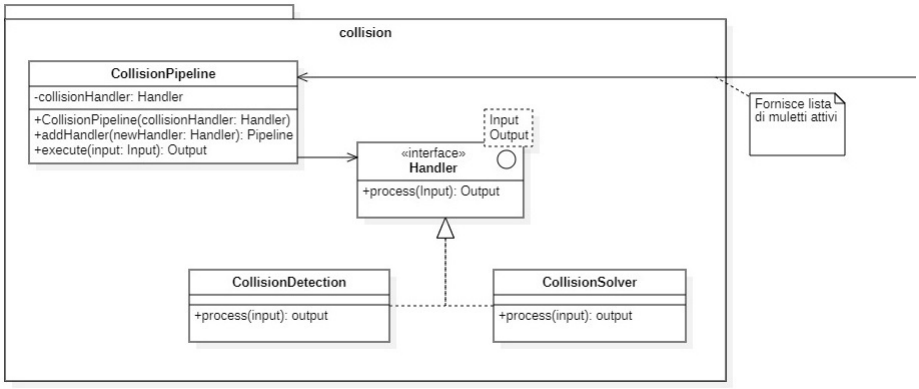

Figura 5.1.5: Visione di dettaglio del package Collision

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:** sulla base delle future mosse di ogni unità, vengono determinate le possibili collisioni;
- **risoluzione:** in caso vengano rilevate, vengono elaborate le mosse, da trasmettere alle unità coinvolte, che impediscano tali collisioni.

Le classi `CollisionDetector` e `CollisionSolver` incapsulano rispettivamente le due funzionalità elencate.

Viene applicato in questo contesto il design $pattern_G$ `Pipeline`[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

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

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 associ i muletti coinvolti e le azioni da intraprendere così da poterle comunicare.

5.1.3 Persistence layer

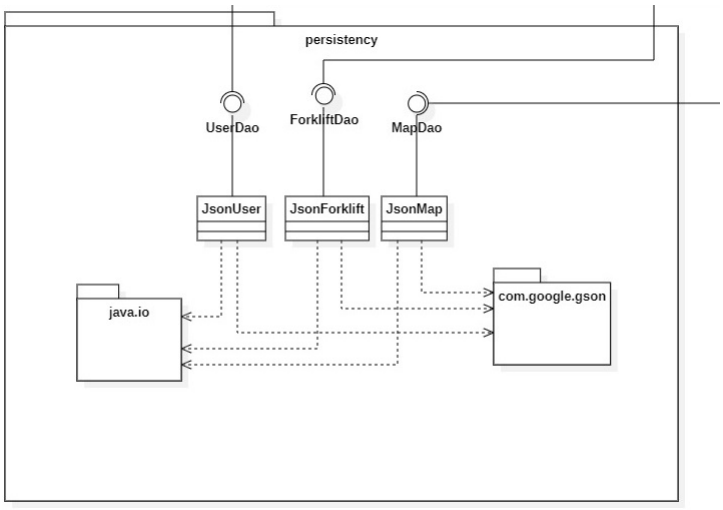

Figura 5.1.6: Visione di dettaglio del Persistence Layer

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.

5.2 Client

5.2.1 Diagramma di classe per l'unità

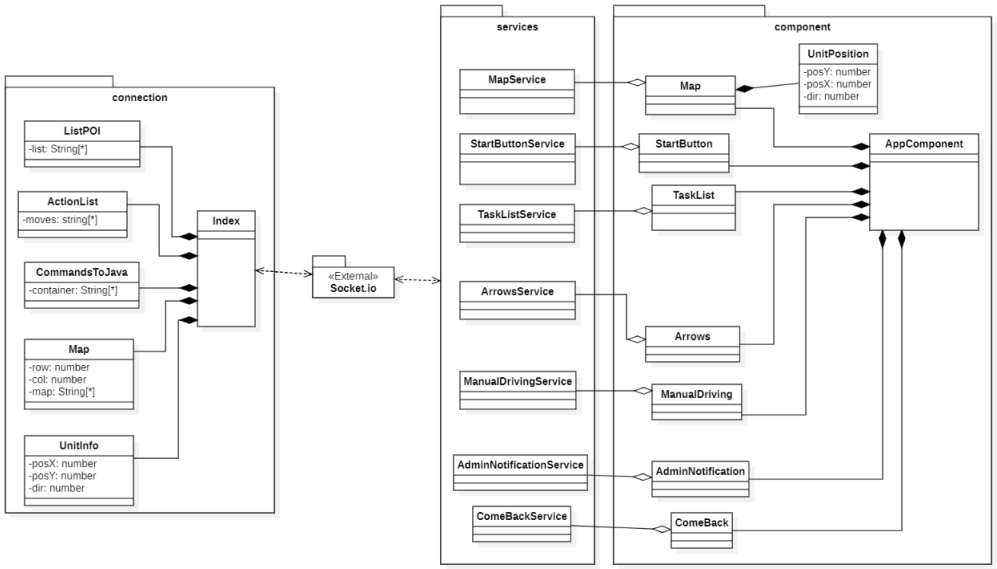

Figura 5.2.1: Diagramma UML_A delle classi per l'unità

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` → visulizza 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 Diagramma di classe per l'admin-manager

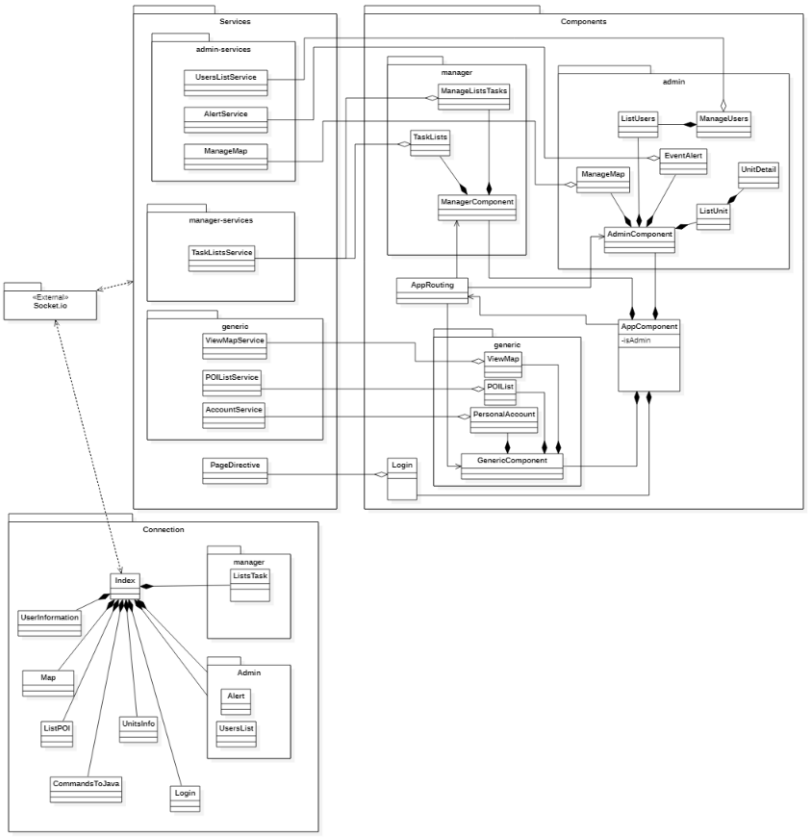

Figura 5.2.2: Diagramma UML_A delle classi per gli admin e i manager

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 del'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.

5.3 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.

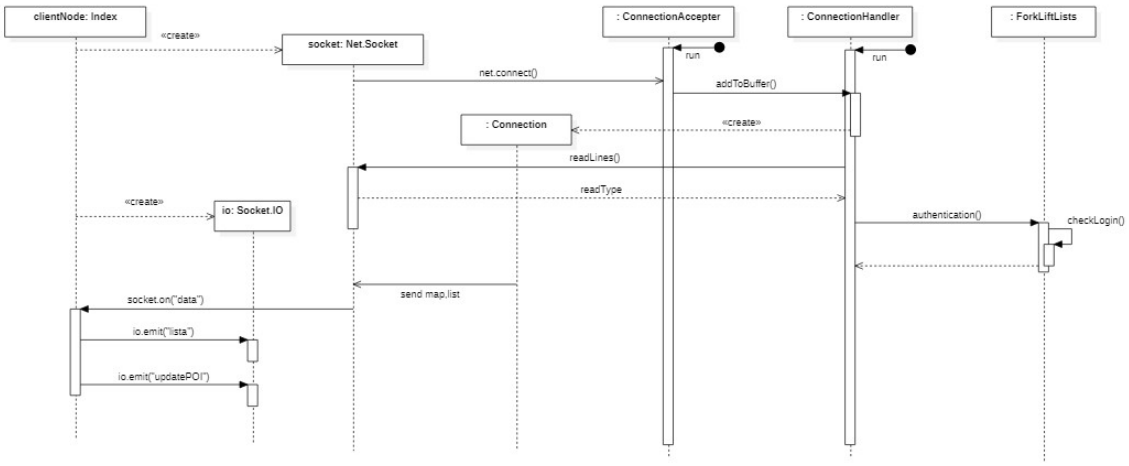

Figura 5.3.1: Visione complessiva della connessione server Java - unità muletto in NodeJS

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).

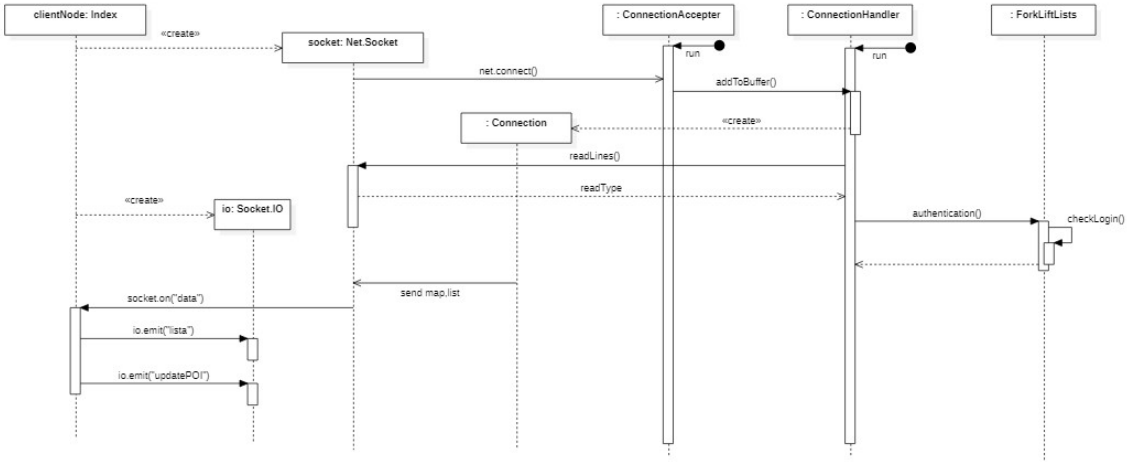

Figura 5.3.2: Esempio di comunicazione tra server e client, attraverso i socket TCP

Questo diagramma rappresenta un esempio di comunicazione tra server e client di tipo muletto. Il server legge i dati inviati dal muletto e invia una risposta, il muletto 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 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.

5.3.2.1.1 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.

5.3.2.2 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 5.3.1: Riepilogo enumerazioni

5.3.2.3 Comandi clients → server

Per i comandi di risposta, controllare i corrispondenti in § 5.3.2.4

FORKLIFTS → SERVER		
Comando	**Descrizione**	**Risposta**
`POS,X,Y,DIR`	Posizione attuale del muletto, considerando la mappa come una matrice: • 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,...`

Tabella 5.3.2: Comandi clients → server | Forklifts

USER generico → SERVER		
Comando	**Descrizione**	**Risposta**
`EDIT,T,PAR`	Modifica dati del proprio profilo • T: NAME/LAST/PWD • PAR: nuova valore per T	–
`LOGOUT`	Richiesta di disconnessione	–

Tabella 5.3.3: Comandi clients → server | User generico

MANAGER → SERVER		
Comando	**Descrizione**	**Risposta**
`ADL,P1,P2,...`	Aggiunge una nuova lista di task$_G$ • P1..: id dei poi che compongono la lista	`ADL,...`
`RML,ID`	Richiede la cancellazione della lista con id=ID	`RML,...`

Tabella 5.3.4: Comandi clients → server | User Manager (Responsabile)

ADMIN → SERVER		
Comando	**Descrizione**	**Risposta**
`MAP,R,C,SEQ`	Nuova planimetria$_G$ • 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[,T,NAME]`	Modifica una cella, la parte tra [ ] è presente solo in caso di POI$_A$ • X e Y riga e colonna della matrice • A: numero rappresentante l'azione da intraprendere, corentemente con CellType Solo nel caso di POI$_A$: • T: tipo di POI$_A$ secondo PoiType • NAME: stringa di caratteri da associare al POI$_A$	`CELL,...`
`ADU,T,NAME,LAST`	aggiunge nuovo utente • T: tipo (ADMIN o MANAGER) • NAME e LAST: rispettivamente nome e cognome	`ADU,...`
`RMU,ID`	Rimuove l'utente con id=ID	`RMU,...`

ADMIN → SERVER		
Comando	**Descrizione**	**Risposta**
`EDU,ID,A,PAR`	Modifica l'utente con id=ID, con • A: azione da intraprendere tra: – 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 5.3.5: Comandi clients → server | User Admin (Amministratore)

5.3.2.4 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$_G$ • 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$_A$ • 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 5.3.6: 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	–
`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 5.3.7: Comandi server → clients | Forklifts (muletti)

SERVER → USER generico		
Comando	**Descrizione**	**Risposta**
`UNI,N,ID1,X1,Y1,D1`	Indica le posizioni di tutti i muletti. Da ID1 in poi si ripete per ogni muletto • 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. Se l'invio di questo fallisce, l'utente viene considerato disconnesso.	–
`LIST,IDF,N,IDP1,IDP2...`	Indica la lista di $task_G$ presa in carico da un muletto • 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 5.3.8: 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 5.3.9: 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 • 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: • N: num totale muletti • IDn, Tn: rispettivamente id e token del muletto n	–
`LISTU,N,UN1,UL1,R1...`	In risposta alla richiesta della lista degli utenti: • N: num totale utenti • UN...: rispettivamente nome, cognome e ruolo	–

Tabella 5.3.10: Comandi server → clients | Utente Admin (amministratore)

6 Estendere PORTACS

In questa sezione verranno riportate tutte quelle informazioni utili ad una semplice e corretta estensione del prodotto PORTACS$_A$. Possono essere estensioni legate sia a nuovi algoritmi e framework$_G$, più efficaci o efficienti, che a nuove categorie di sotto-sistemi di PORTACS$_A$.

6.1 Algoritmo alternativo per 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.

6.2 Introdurre nuove tipologie di utenti

6.2.1 Lato server

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 ProcessComunication, che si occupa di gestire la comunicazione, in particolare ricevere i messagi, eseguire quello che viene richiesto e inviare i dati necessari.

6.2.2 Lato client

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`.

6.3 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*.

6.4 Modificare handler nell'algoritmo di gestione delle collisioni

La struttura fornita dal design pattern$_G$ *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$_G$ 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.