>Three Way Milkshake_

Manuale Manutentore

Three Way Milkshake - Progetto "PORTACS"

threewaymilkshake@gmail.com

Versione	0.0.1
Stato	Non approvato
Uso	Esterno
Approvazione	———
Redazione	———
Verifica	———
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	Redazione	Data red.	Verifica	Data ver.
0.4.0	Stesura § 6.0	Tessari Andrea	2021-04-23	———	———
0.3.0	Stesura § 3.0 e § 3.1	Tessari Andrea	2021-04-23	———	———
0.2.1	Incremento § 5.2	Zuccolo Giada	2021-04-22	———	———
0.2.0	Stesura § 5.2	Tessari Andrea	2021-04-22	———	———
0.1.1	Stesura § 2.2	Chiarello Sofia	2021-04-22	———	———
0.1.0	Stesura § 1	Tessari Andrea	2021-04-22	———	———
0.0.1	Impaginazione	De Renzis Simone	2021-04-15	———	———

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, 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. 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-princip
- 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/;
- Docker:
  https://www.docker.com/.

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-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. 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 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 il la serializzazione e deserielizzazione dei file Json della persistenza.
  - Documentazione: https://github.com/google/gson/blob/master/UserGuide.md
- **JUnit**: framework di unit testing per il linguaggio di programmazione Java.
  - Versione utilizzata: 5.7
  - Documentazione: https://junit.org/junit5/docs/current/user-guide/
- **Mockito**: framework open source di test per il linguaggio di programmazione Java. Il framework 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**
  Node.js è un 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**
  HTML è un linguaggio markup per la strutturazione di pagine web. Viene utilizzato insieme a Angular per la costruzione della struttura della web app.

- **CSS**
  CSS è un 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 conte-nuti 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 contem-poraneamente anche il riutilizzo di codice ed una sua più facile manutenzione. Viene utilizzato insieme ad Angular per la stilizzazione degli elementi HTML.

- **Typescript**
  TypeScript è un 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**
  Angular è un 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 azione da parte dell'utente.

  – Versione utilizzata: 11.2.0
  – Link per installazione: https://angular.io/guide/setup-local

- **PrimeNG**
  PrimeNG è una libreria per Angular per customizzare i componenti così da creare un'interfaccia utente più accattivante.

  – Link per installazione: https://primefaces.org/primeng/showcase/#/setup

2.3 Versionamento e Continuous Integration

- **GitHub**
- **GitHub Action**

3 Setup

Il progetto PORTACS è composto da 3 container Docker differenti, uno rappresentante il backend, un altro creato per l'operatore dell'unità e l'ultimo ideato per l'admin o i manager. Il container per l'unità sarà installato su un muletto, quello per l'admin o i manager su un personal computer e quello rappresentante il backend su un server. I contenitori del client funzioneranno su un browser che supporti bene HTML e CSS.

3.1 Requisiti di sistema

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

3.1.1 Requisiti Hardware

- Client -> unità:
  - CPU single-core o maggiore;
  - Memoria Ram >= 2GB;
  - definizione schermo >= 640 x 480 (Standard Definition);
  - connessione internet con bassi tempi di risposta.
- Client -> admin o responsabile:
  - CPU dual-core o maggiore;
  - Memoria Ram >= 3GB;
  - definizione schermo >= 1280 x 720 pixel (High Definition);
  - connessione internet con bassi tempi di risposta.
- Server:
  - CPU dual-core o maggiore;
  - Memoria Ram >= 4GB;
  - connessione internet con bassi tempi di risposta.

3.1.2 Requisiti Software

- Client:
  - Docker.
- Server:
  - Docker.

3.2 Installazione

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 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 di unit testing per la programmazione java. Junit viene utilizzato per i test di unità relativi al codice in Java. Per effettuare i test basterà eseguire il comando ./gradlew test.

4.3.2 Spring

Framework utilizzato per la risoluzione delle dipendenze e come supporto durante la scrittura degli unit test.

4.4 Testing lato client

4.4.1 Jasmine

Jasmine è un framework 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. Karma e Jasmin vengono utilizzati in quanto ramework di default per unit testing di Angular CLI.

4.4.3 Mocha

Mocha è un framework 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.

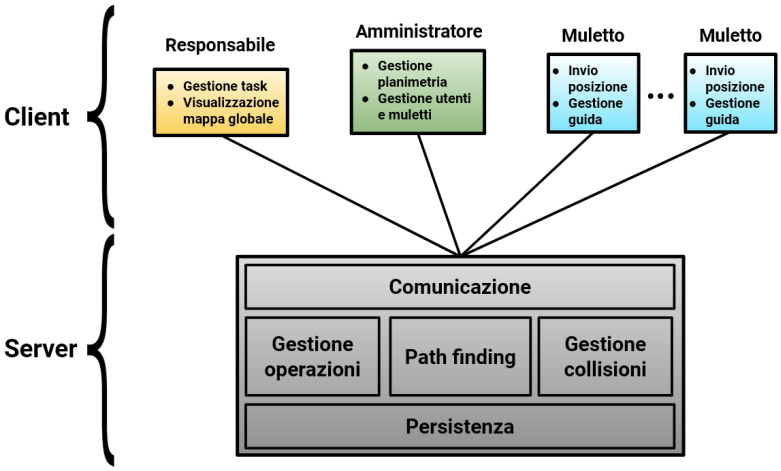

Figura 5.0.1: Rappresentazione ad alto livello dell'architettura del software

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**

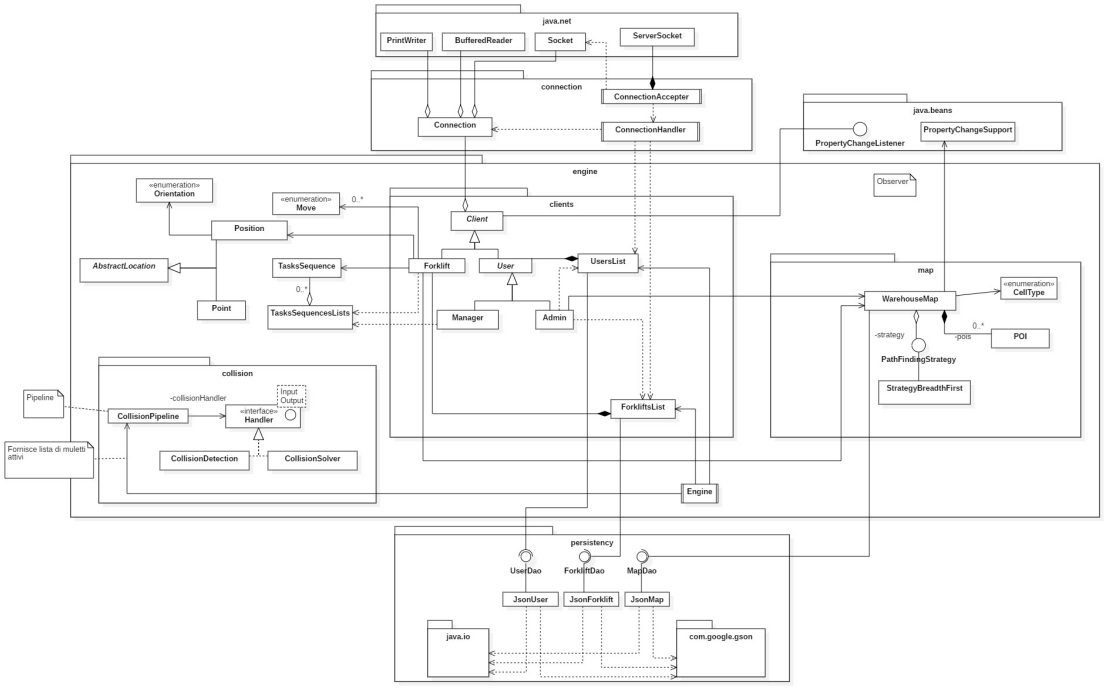

Figura 5.1.1: Visione complessiva dell'architettura del server

Le sezioni che seguono illustrano la struttura di ogni layer.

5.1.1 Communication layer

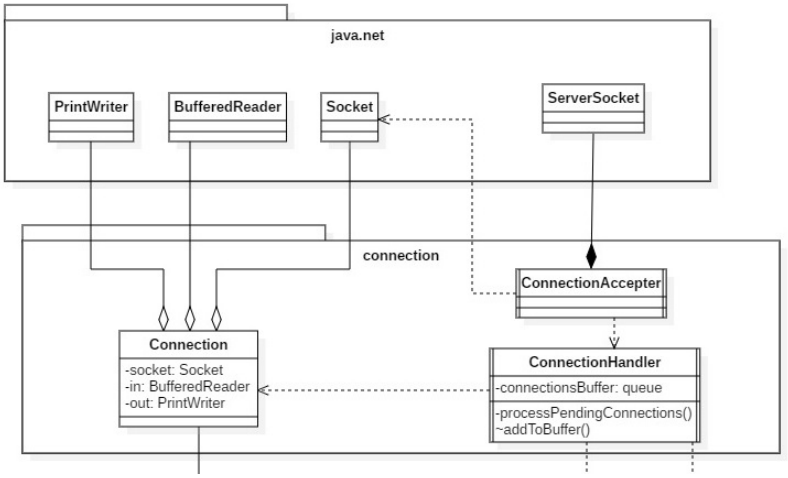

Figura 5.1.2: Visione di dettaglio del Communication Layer

5.1.2 Business layer

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

- gestione della mappa e path finding;
- gestione dell'autenticazione dei client;
- gestione delle tasks e dei POI;
- rilevazione e gestione 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

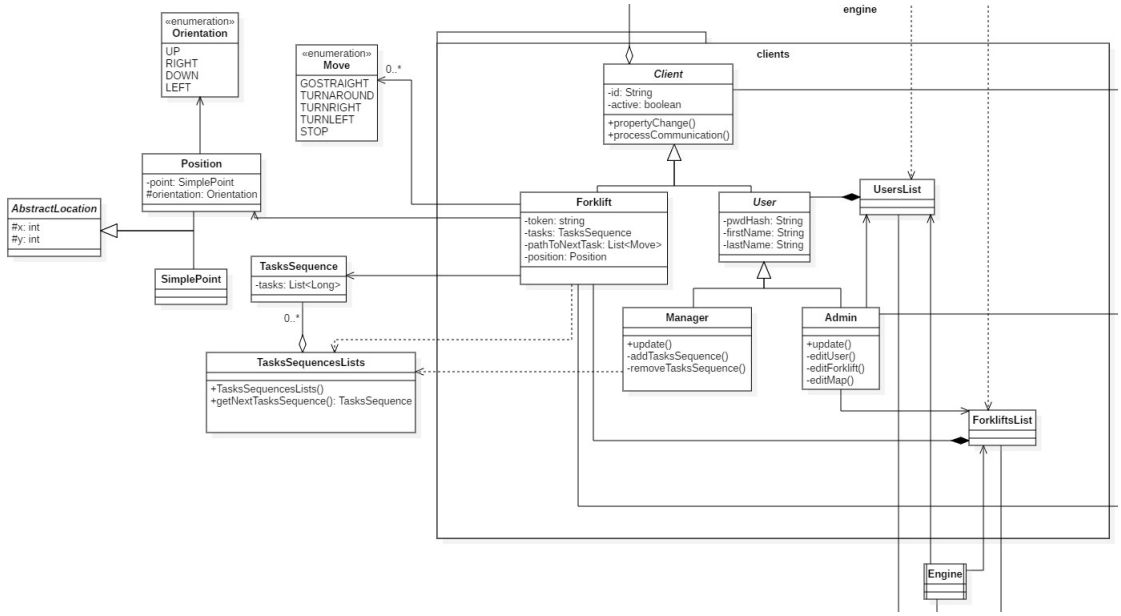

Figura 5.1.3: Visione di dettaglio del package Clients

Il server conserva nelle classi `UsersList` e `ForkliftList` le liste degli utenti e dei muletti (client) connessi con i relativi token di autenticazione. In particolare, 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).

I `Forklift` di caratterizzano dagli attributi:

- `Position`: rappresenta la posizione e orientamento attuali del muletto nella mappa;
- `TaskSequence`: una sequenza di task da compiere;

- `Move`: una lista di mosse atte a raggiungere il prossimo POI (e quindi evadere la prossima task).

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 `UsersList` e `ForkliftList` con i seguenti obiettivi:

- ricevere le nuove posizioni dai muletti;
- inviare le nuove informazioni agli utenti per la visualizzazione nel monitor real-time;
- processare eventuali altre richieste (calcolo percorso, aggiunta task, modifica mappa).

Dopodichè la `ForkliftList` viene utilizzata dal modulo di rilevazione gestione delle collisioni.

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

5.1.2.2 Mappa

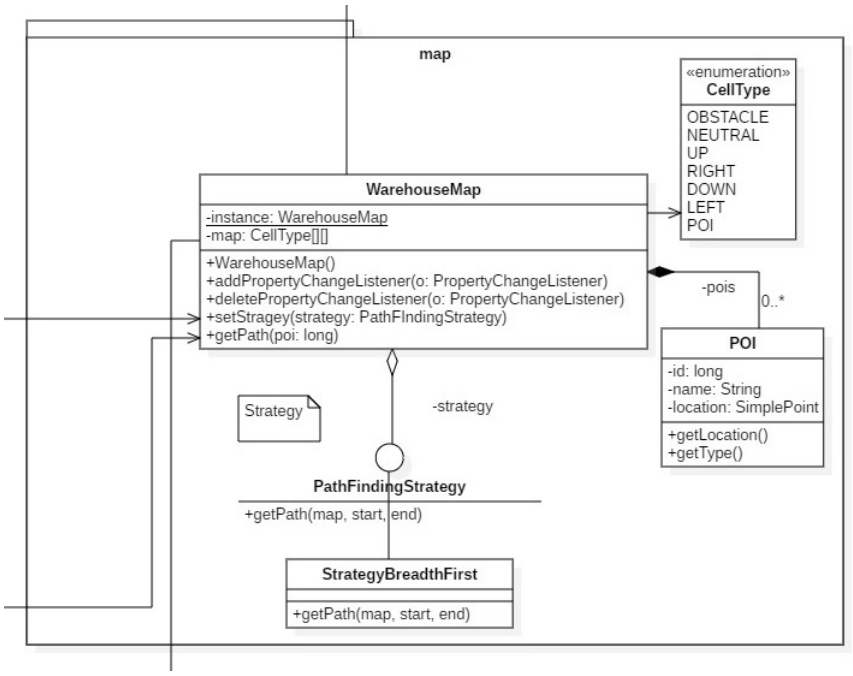

Figura 5.1.4: Visione di dettaglio del package Map

La classe `WarehouseMap` contiene la rappresentazione della planimetria 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, e per ognuno la relativa locazione. Si osserva l'applicazione di alcuni design pattern:

- **observer**: tramite la libreria `PropertyChangeSupport` e `PropertyChangeListener` di Java 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 comunicare ai client esterni le modifiche, e possano essere aggiornate 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

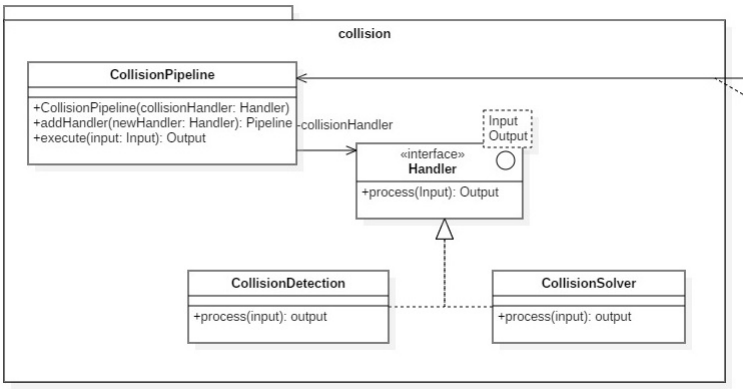

Figura 5.1.5: Visione di dettaglio del package Collision

5.1.3 Persistence layer

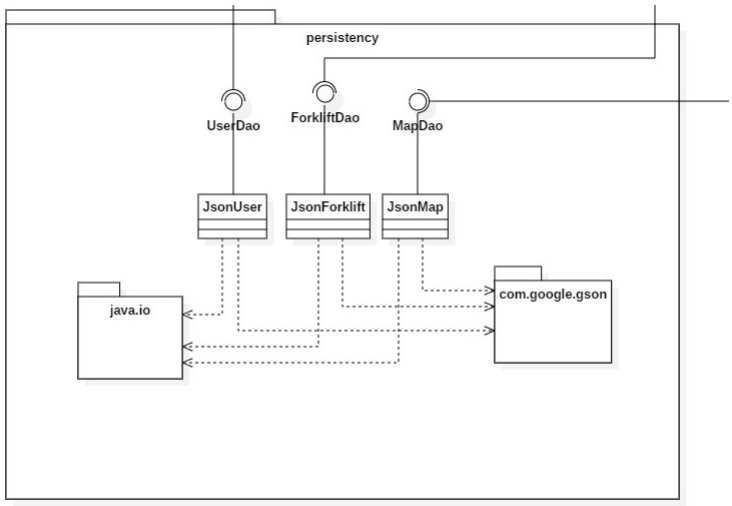

Figura 5.1.6: Visione di dettaglio del Persistence Layer

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

- le credenziali di autenticazione degli utenti;
- 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

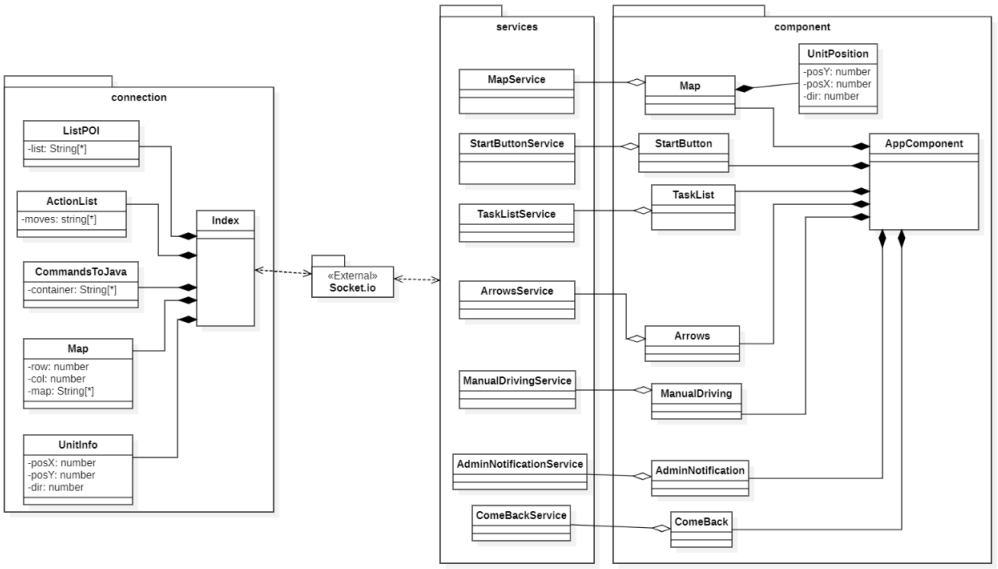

Figura 5.2.1: Diagramma UML delle classi per l'unità

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 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

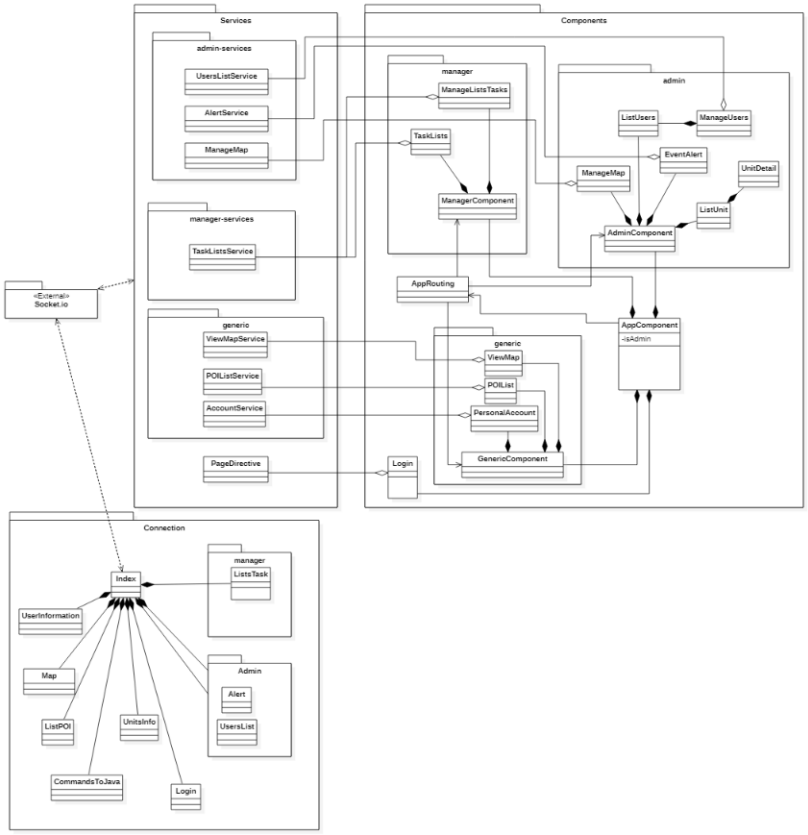

Figura 5.2.2: Diagramma UML delle classi per gli admin e i manager

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 amnagere 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 e le caratteristiche della mappa;
- manager:
  - TaskLists -> visualizzazione delle liste di task;
  - MangeListsTask -> aggiunta, modifica e rimozione di liste di task;
- generic (sia per admin che per manager):
  - ViewMap e POIList -> visualizzazione in real time di tutte le unità nel magazzino.

5.3 Comunicazione

5.3.1 Diagrammi di sequenza

5.3.2 Protocollo di comunicazione

6 Estendere PORTACS

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

6.1 Algoritmo alternativo per il path finding

6.2 Introdurre nuove tipologie di utenti

6.2.1 Lato server

6.2.2 Lato client

6.3 Implementare tipi di persistenza alternativi

6.4 Modificare handler nell'algoritmo di gestione delle collisioni