Relazione Tecnica - Damose

Frontespizio

- Progetto: Damose - Rome Transit Tracker
- Sviluppatore: Giorgio Caruso

1. Obiettivi e funzionalita implementate

1.1 Obiettivi

L'obiettivo del progetto e realizzare una applicazione desktop Java per la consultazione del trasporto pubblico di Roma con:

- Modalità offline su GTFS statico;
- Modalità online con GTFS Realtime;
- Mappa interattiva per visualizzare mezzi e linee;
- Ricerca fermate/linee, preferiti, previsioni arrivi.

1.2 Funzionalità implementate

Funzionalità principali presenti:

- Ricerca fermate per nome/codice.
- Ricerca linee per numero/nome.
- Visualizzazione percorso linea su mappa.
- Visualizzazione mezzi in tempo reale su mappa con refresh periodico.
- Filtri per linea e direzione.
- Arrivi fermata da statico + realtime.
- Preferiti per fermate e linee.
- Login/registrazione utente con persistenza locale.
- Switch online/offline con fallback.
- Dashboard sintetica qualità servizio.

2. Architettura e scelte progettuali

2.1 Architettura

Il progetto e strutturato in livelli:

- `damose.view`: UI Swing e componenti mappa.
- `damose.controller`: orchestrazione casi d'uso.

- `damose.service`: logica applicativa (arrivi, realtime, preferiti, qualità).
- `damose.data.loader` e `damose.data.mapper`: ingestione GTFS statico e matching realtime.
- `damose.model`: entità dominio.
- `damose.database`: persistenza utenti/sessione.

2.2 Pattern e principi usati

- MVC: separazione fra View, Controller, Model.
- Service Layer: logica applicativa isolata da UI.
- Loader/Mapper: separazione parsing e trasformazione dati.
- Single responsibility: classi focalizzate (es. `RouteService`, `ArrivalService`).
- Threading controllato: aggiornamenti UI con `SwingUtilities.invokeLater`.

2.3 Motivazioni

Le scelte privilegiano:

- Manutenibilità (modularità e responsabilità chiare);
- robustezza in modalità offline/online;
- Estendibilità (nuove metriche o nuove sorgenti feed).

3. Dati GTFS statici e realtime

3.1 Pipeline dati

- GTFS statico caricato da risorse locali (`stops`, `trips`, `stop_times`, `calendar_dates`, `routes`).
- GTFS Realtime acquisito da endpoint Roma Mobilita:
  - `vehicle_positions`
  - `trip_updates`
- Parsing protobuf e normalizzazione id trip/route.
- Mappa aggiornata periodicamente in online.

3.2 Modalità offline/online

- Offline: arrivi e posizione stimata da schedule statica.
- Online: arrivi aggiornati con trip updates + posizione mezzi realtime.
- Fallback: gestione stato connessione e passaggio operativo in assenza feed.

4. UI e UX

- Interfaccia desktop Swing con tema scuro.
- Mappa interattiva con zoom/pan.
- Pannelli overlay:
  - risultati ricerca;
  - arrivi fermata;
  - pannello linea con fermate, veicoli e cambio direzione.
- Pulsanti rapidi per ricerca, preferiti, toggle visibilità mezzi, stato connessione.

5. Testing e qualità codice

- Build e dipendenze con Maven.
- Test unitari presenti in `src/test/java` su modelli, mapper e servizi chiave.
- Jar eseguibile configurato con `maven-shade-plugin`.
- Documentazione API generata in `docs/javadoc`.

Nota: nello stato corrente del repository alcuni test hanno import legacy da allineare dopo la riorganizzazione pacchetti.

6. Copertura criteri MDP (sintesi)

- Livello base: sostanzialmente coperto.
- Livello intermedio: coperto nelle funzionalità principali (realtime, preferiti, switch online/offline, Maven, jar).
- Livello avanzato: parzialmente coperto (autenticazione presente; metriche avanzate e ETA intelligente da completare).

Gap principali da completare:

- metriche avanzate qualità servizio (corse mancanti/deviate/interrotte storicizzate per linea);
- modello ETA intelligente basato su storico ritardi;
- posti disponibili a bordo (se dato presente nel feed).

7. Paradigmi richiesti

7.1 Programmazione generica

Uso esteso di collezioni tipizzate (`List<Stop>`, `Map<String, Trip>`, ecc.) e API con tipi dominio forti.

7.2 Programmazione funzionale

Uso di lambda/callback in listener UI, scheduler e binding eventi.

7.3 Stream API

Utilizzo stream per mapping, filtri e ordinamenti nella composizione dati lato controller/servizi.

8. Risorse esterne

- OpenStreetMap tramite JXMapViewer2.
- GTFS Realtime bindings + protobuf Java.
- FlatLaf per tema UI.
- SQLite JDBC per persistenza locale utenti.
- Open data Roma Mobilita per feed realtime.

9. Contributi e ruoli:

- Sviluppatore: Giorgio Caruso: backend, frontend, testing, integrazione.

10. Uso strumenti AI

Sono stati usati strumenti AI di supporto alla produttività per:

- proposta di fix e verifiche veloci.
- stesura/riordino documentazione tecnica.
- Inizializzazione struttura classe generica.

Tutte le modifiche sono state verificate, adattate e validate manualmente nello specifico contesto del progetto. Essendo il mio primo progetto mai fatto e soprattutto il mio primo lavoro in Java mi sono potuto facilitare molto con l'utilizzo di IA per conoscenza generale di utilizzo di metodi e sintassi e analisi dei dati con il quale stavo lavorando. Sono molto contento di come son riuscito a lavorare utilizzando questo potente strumento.

11. Conclusioni e commento personale

Il progetto raggiunge una base solida per i requisiti core del corso, con un'app desktop funzionante offline/online, mappa interattiva, ricerca e preferiti. Le estensioni principali per massimizzare la fascia alta riguardano quality analytics avanzati e ETA intelligente

su storico ritardi.

Penso di aver svolto un buon lavoro e soprattutto mi ha appassionato portare avanti un progetto tutto da solo riuscendo a mettere in pratica quello che volevo.

Senza l'IA il progetto mi sarebbe stato molto più difficile soprattutto con un linguaggio verboso come Java, infatti mi ha aiutato molto iniziare con un architettura di una classe disegnata dall'IA e modificare bug e accortezze di mia mano.