Rome Transit Tracker (Damose)

[Edit: potete utilizzare come template per il progetto e l'interfacciamento con le librerie questo starter code]

1. Introduzione

L'obiettivo del progetto è sviluppare un'applicazione standalone per desktop che consenta agli utenti di visualizzare in tempo reale la posizione degli autobus nella città di Roma. L'applicazione dovrà funzionare sia online che offline, utilizzando rispettivamente i dati in tempo reale e degli orari statici degli autobus.

L'applicazione dovrà consentire agli utenti di:

- Consultare le fermate e le linee disponibili.
- Salvare fermate e linee preferite.
- Visualizzare la posizione attuale degli autobus su una mappa.
- Predire gli arrivi ad una fermata basandosi sia sui dati in tempo reale che su quelli statici.

2. Funzionamento dell'applicazione completa

2.1. Fermate e linee

- Ricerca di una fermata per nome o codice e visualizzazione della posizione.
- Visualizzazione della lista delle linee che servono una fermata.
- Ricerca di una linea per numero o nome e visualizzazione del percorso.
- Possibilità di salvare fermate e linee preferite.

2.2. Mappa e visualizzazione dei bus

- Visualizzazione di una mappa interattiva, con:
    - Posizione attuale stimata dei bus.
    - Aggiornamenti automatici ogni 30 secondi (se online).
    - Filtri per selezionare solo una linea o una zona specifica.
    - Navigazione tramite zoom e pan.

2.3. Previsioni di arrivo

- Se online, previsioni aggiornate in tempo reale basate su GTFS Realtime.

- Se offline, previsioni basate sulla schedule statica GTFS.

2.4. Autenticazione utente

- Registrazione ed autenticazione di utenti.
- Gli utenti autenticati possono:
    - Salvare preferiti (linee e fermate).
    - Personalizzare le impostazioni dell'app.
- Il salvataggio delle preferenze avviene in locale per garantire il funzionamento offline.

2.5. Gestione dello stato della connessione

- Se il feed GTFS realtime non è disponibile, il sistema deve:
    - Mostrare un messaggio di avviso.
    - Passare automaticamente alla modalità offline.

2.6. Statistiche della qualità del servizio

- Archiviazione delle corse mancanti, interrotte, deviate, ed in ritardo, confrontando le corse programmate (GTFS statico) con i dati reali (GTFS realtime).
- Computo della percentuale di corse puntuali, ritardate o saltate per linea, sulla base dello storico.
- Aggiornamento delle previsioni d'arrivo sulla base dei ritardi tipici.
- Visualizzazione delle metriche della qualità di servizio nella GUI

3. Feature richieste

3.1. Livello base (18-23/30)

1 programmatore

- Funzionamento offline, con dati statici GTFS.
- Visualizzazione e ricerca delle fermate, che mostri le prossime linee che vi ci fermeranno ed i corrispondenti orari di arrivo.
- Visualizzazione e ricerca linee, che mostri la fermata attuale per ogni mezzo della linea.
- Predizione dell'orario di arrivo di una linea ad una fermata basata sulla schedule statica.
- Mappa di visualizzazione della posizione dei mezzi sulla base della schedule statica (non interattiva e senza aggiornamenti in tempo reale), che mostri il numero/codice della linea e la direzione del mezzo.
- Gestione differenziata delle diverse tipologie di mezzi (bus, tram, etc.).

2–4 programmatori; come sopra, più

- Sviluppo del progetto tramite git.
- Mappa interattiva (zoom in/out) statica.

3.2. Livello intermedio (24-27/30)

1 programmatore

- Aggiornamento in tempo reale della posizione dei bus (se online), sia sulla mappa che nei risultati di ricerca.
- Possibilità di salvare preferiti (linee e fermate).
- Predizioni di arrivo usando i dati in tempo reale.
- Switch automatico tra online e offline.

2–4 programmatori; come sopra, più

- Testing unitario per validare il funzionamento del sistema.
- Gestione delle dipendenze tramite maven o gradle.
- Creazione di un Jar eseguibile.
- Visualizzazione della quantità di posti disponibili a bordo, quando disponibili.

3.3. Livello avanzato (28-30/30)

1 programmatore

- Autenticazione utente e gestione personalizzata dei preferiti.
- Testing unitario per validare il funzionamento del sistema.

2–4 programmatori; come sopra, più

- Monitoraggio della qualità del servizio.
- Dashboard di visualizzazione della qualità del servizio.
- Previsione d'arrivo "intelligente" della linea, dell'orario, e dei ritardi recenti di altre linee"intelligente".

4. Valutazione

È possibile svolgere il progetto singolarmente o in un gruppi di fino a quattro sviluppatori. Lo sviluppo in team richiede il completamento di feature aggiuntive rispetto allo sviluppatore singolo. È possibile ottenere il punteggio massimo sia come sviluppatori singoli che in gruppo. La valutazione è individuale, anche nel caso di sviluppo in gruppo. Le feature richieste sono divise in tre fasce, che corrispondono a tre range di voto: base (18-23), intermedio (24–26), ed avanzato (27–30). Per esempio, per ottenere un voto nel range 27–30 è necessario implementare almeno parte delle feature definite nel livello avanzato, in aggiunta a tutte quelle delle fasce precedenti. Il voto finale, all'interno di ciascun range di voto, verrà deciso sulla base dell'implementazione, della documentazione del codice, del diagramma delle classi, e della relazione. In particolare, verranno valutati:

- La corretta applicazione dei principi OOP e design pattern.
- La completezza e stabilità delle feature implementate.
- La qualità del codice (leggibilità, modularità, manutenibilità, scalabilità).
- L'usabilità, creatività, e funzionalità dell'implementazione al di là delle specifiche.

5. Codice di condotta

Mi aspetto che ciò che consegnate sia frutto del vostro lavoro individuale. È difficile dare linee guida specifiche sul plagio, ma come "rule of thumb": se non è stato digitato da voi, non è frutto del vostro lavoro. È concesso scambiare idee al di fuori del gruppo di lavoro, ma non condividere e commentare codice. È concesso avvalersi di risorse e strumenti che vi aiutino nello sviluppo del codice, ricercare informazioni online, usare librerie esterne, ed avvalersi di strumenti di AI, ma la relazione e la documentazione del codice devono segnalare puntualmente quali porzioni di codice sono contributi originali. La valutazione verterà sulla progettazione, il codice, e le funzionalità sviluppate da voi. Consegne considerate plagio o in altro modo non frutto del lavoro dello studente verranno registrate come esami falliti e segnalate al comitato etico di facoltà.

6. Risorse

- Documentazione degli standard (formato e protocollo di accesso) per le informazioni sul trasporto pubblico, sia statiche che in tempo reale
- Open data e controllo satellitare del trasporto pubblico per la mobilità di Roma
- Una libreria utile per la visualizzazione di mappe openstreetmap
- Una libreria utile per il parsing di dati real time

7. Istruzioni per lo sviluppo

- Il progetto deve essere sviluppato in Java 17+
- L'interfaccia utente dev'essere sviluppata con JavaFX o Swing
- Ogni membro del team deve assumere uno o più dei seguenti ruoli dall'inizio del progetto:
    - Sviluppatore backend: si occupa dell'integrazione della logica di programma con i dati GTFS.
    - Sviluppatore frontend: implementa l'interfaccia utente e la visualizzazione della mappa.
    - Tester: scrive test per garantire il corretto funzionamento del programma.
- Le chiamate alle API di open data Roma devono essere limitate a 1 ogni 30 secondi durante l'esecuzione dell'applicazione. Durante lo sviluppo, è richiesto il caching dei dati da una chiamata precedente, o la simulazione dei dati.

8. Istruzioni di consegna

- Ogni team o sviluppatore singolo dovrà consegnare:
    - Codice sorgente ben documentato. Dev'essere distribuito come progetto eclipse in formato ZIP, chiamato "Damose_cognome1[_cognome2, …].zip" con tutte le cartelle relative a codice sorgente e risorse necessarie all'esecuzione, incluse le librerie ed i file di configurazione. La classe "Main" deve contenere il metodo main del programma. Il progetto deve contenere una cartella "docs" contenente la documentazione completa generata con javadoc.
    - Una relazione tecnica, con spiegazione delle scelte progettuali in formato pdf. Ciascun gruppo consegnerà una sola relazione collegiale, nella quale una sezione deve specificare i ruoli ed i contributi dei singoli membri, ed una valutazione tra pari di tali contributi. La prima pagina deve indicare nome, cognome, e numero di matricola di ciascun membro del gruppo. Una sezione

deve dichiarare quali feature siano state implementate. Una sezione deve dichiarare le risorse esterne utilizzate e la loro integrazione (se presenti). La relazione deve discutere le decisioni di progettazione, i design pattern adottati, e l'applicazione di paradigmi di programmazione generica, funzionale, e stream.
    - il diagramma delle classi in formato PDF. Potete utilizzare strumenti di disegno come draw.io, Google Slides, PowerPoint, etc.
- Durante la presentazione, i team dovranno dimostrare:
    - Il funzionamento dell'app in modalità offline e online.
    - La qualità dell'interfaccia utente e della struttura del codice.
    - La distribuzione dei ruoli nel team di sviluppo.

Note