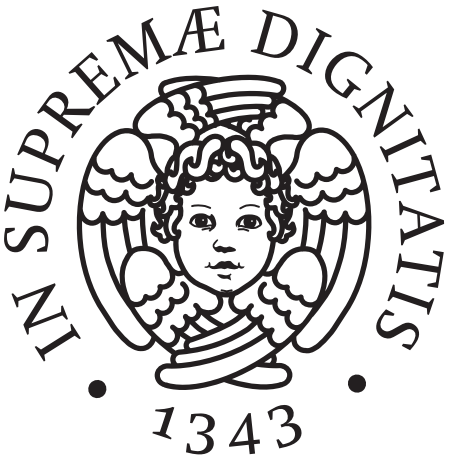


UNIVERSITÀ DI PISA

Progetto Java

studente Alessandro Fossari

per il corso di programmazione avanzata, a.a. 24/25

Sommario

Caso d’uso

Il progetto fa riferimento ad un’azienda di trasporti, Fast Transport, e immagina una possibile architettura che consente ai dipendenti di consultare il database aziendale e, allo stesso tempo, preservare i loro dati personali.

Il database è collocato su un processo server cha fa uso del framework Spring.

Per implementare quanto detto, i dipendenti che accedono al database si suddividono in due categorie:

- Gli *admin*: sono gli amministratori del database. Ad essi è consentito prendere visione di tutti i viaggi programmati, di tutti gli utenti registrati, dei mezzi a disposizione e delle diverse sedi dell’attività.
- I *worker*: sono i dipendenti *manuali* dell’attività, ovvero chi effettua le consegne. Ad essi è concessa la sola visualizzazione dei viaggi programmati che li riguardano ed è per loro impossibile prendere visione degli impegni altrui.

Il database

Il database aziendale si compone di quattro tabelle:

- Dipendente (username, password, ruolo, sessionToken): descrive i dipendenti dell’azienda. Banalmente, *username* e *password* sono utilizzati durante le procedure

di registrazione e login al servizio; *ruolo* specifica a quale classe di dipendenti l'utente appartiene (admin o worker). *SessionToken* è invece un campo utilizzato per rendere la comunicazione tra client e server in qualche modo più sicura: anziché comunicare al server il proprio nome utente e la propria password durante le normali operazioni, quest'ultimo genera in risposta al login una *chiave di sessione*, appunto il sessionToken, che l'utente può utilizzare per farsi riconoscere.

- Veicolo (nome): memorizza tutti i veicoli a disposizione dell'azienda.
- SedeLogistica (nome): memorizza tutte le possibili partenze e destinazioni dei viaggi.
- Viaggio (dipendente, veicolo, data, partenza, arrivo, carico): memorizza tutti i viaggi previsti. Un viaggio è identificato dal *dipendente* che lo svolge, dal *veicolo* utilizzato, dalla data in cui viene svolto e dalle sedi di *partenza* ed *arrivo*. L'ultimo attributo, *carico*, è un file memorizzato nel database: esso contiene un documento json che descrive il carico del viaggio e che può essere scaricato dai dipendenti per prenderne nota.

La struttura della comunicazione

Per comunicare, le due entità client e server fanno uso (eccetto che nel caso di */download*) una classe apposita, *DynamicMessage*. Quest'ultima può contenere un array di *String* e mette a disposizione dei metodi per la sua serializzazione e deserializzazione.

Le informazioni scambiate sono serializzate come stringhe *Json*, preferendole a quelle *XML* dato il supporto nativo di Spring per il primo dei due formati.

L'unico caso in cui la comunicazione non fa uso di un'istanza di *DynamicMessage* è nella risposta del server alla richiesta POST /download, nella quale i dati sono trasmessi come una semplice String.

Il controllo sugli accessi

Come già anticipato, sono previste due classi di utenti ed è dunque necessario implementare un meccanismo di controllo degli accessi per evitare che quelli con *capabilities* ridotte possano effettuare azioni indesiderate.

Questo meccanismo è implementato attraverso diverse tecniche:

- Il meccanismo dei *connectionToken* implica che un malintenzionato debba conoscere il token di sessione di un admin per fingersi lui e poter modificare il database.

- Gli API che richiedono la verifica dei privilegi del client sono accessibili da percorsi del tipo "…/users/{nomeUtente}/nome_API": una tale struttura consente al server di cercare nel database il dipendente *nomeUtente*, accedere al suo ruolo per verificarne i privilegi e di confrontare il *connectionToken* in suo possesso con quello appena ricevuto per confermarne l'identità.

Il server

Come anticipato il server fa uso del framework Spring per interfacciarsi con i client e per interagire con il database.

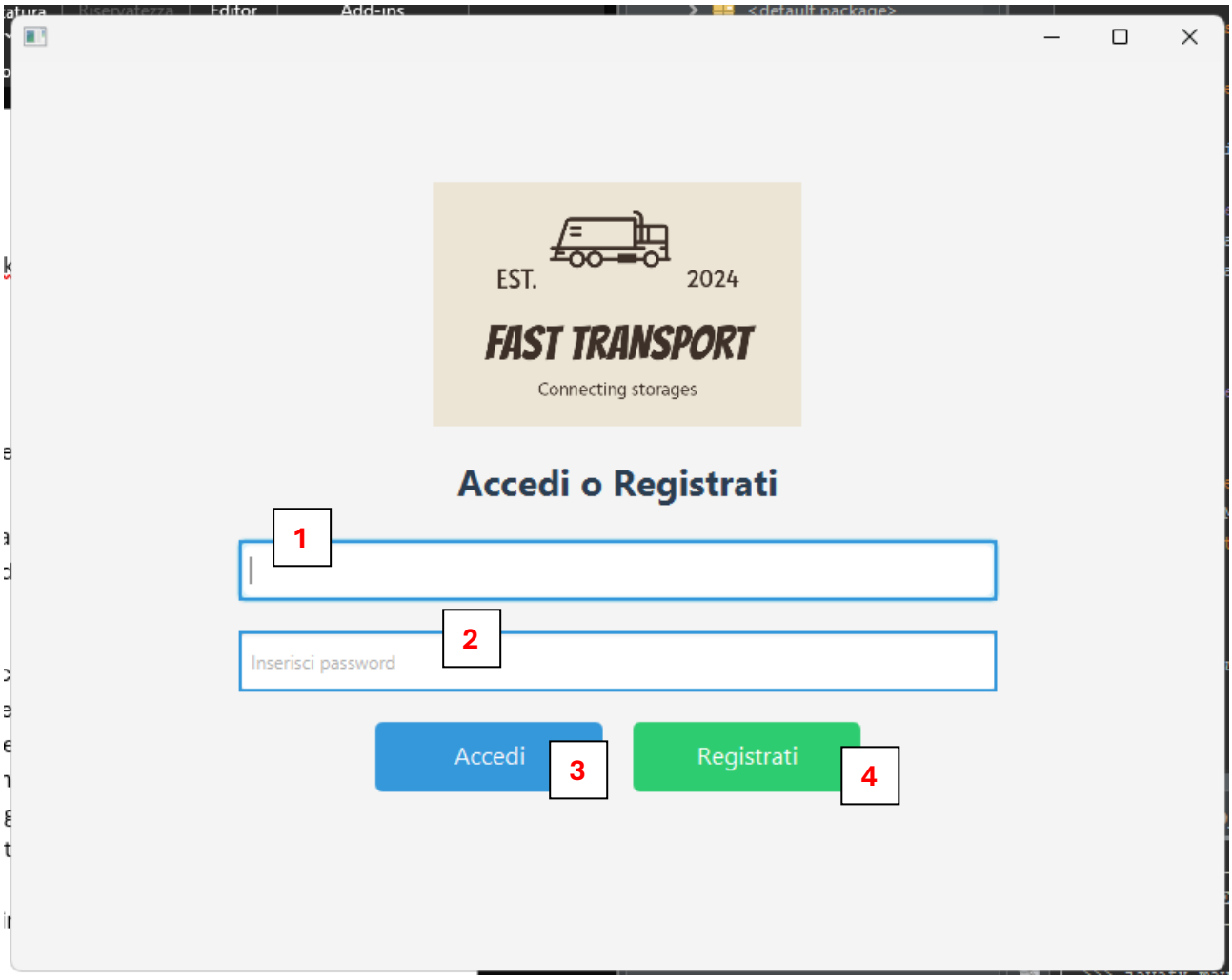
Gli API messi a disposizione sono tutti di tipo POST, data la necessità di implementare i meccanismi detti prima o dato il volume, non sempre ridotto, dei dati scambiati.

Ecco una lista dei diversi API:

- ***POST /register***: consente all'utente di registrarsi come worker o come admin.
- ***POST /login***: permette all'utente di accedere al servizio utilizzando le proprie credenziali e di ricevere il suo token per il resto della comunicazione.
- ***POST /users/{username}/view***: ritorna al chiamante tutti i viaggi previsti nel mese selezionato. Ai worker vengono ritornati solo i viaggi che li vedono partecipi.
- ***POST /users/{username}/possibilities***: ritorna tutti i dipendenti, le sedi e i veicoli memorizzati nel database.
- ***POST /users/{username}/insert***: permette l'inserimento nel database di nuovi veicoli, viaggi e sedi.
- ***POST /users/{username}/download***: dato un viaggio, risponde con il contenuto del suo documento di carico. Il file viene salvato in "resources/documentiDiCarico".
- ***POST /users/{username}/remove***: consente la rimozione dal database di un viaggio.

Il client

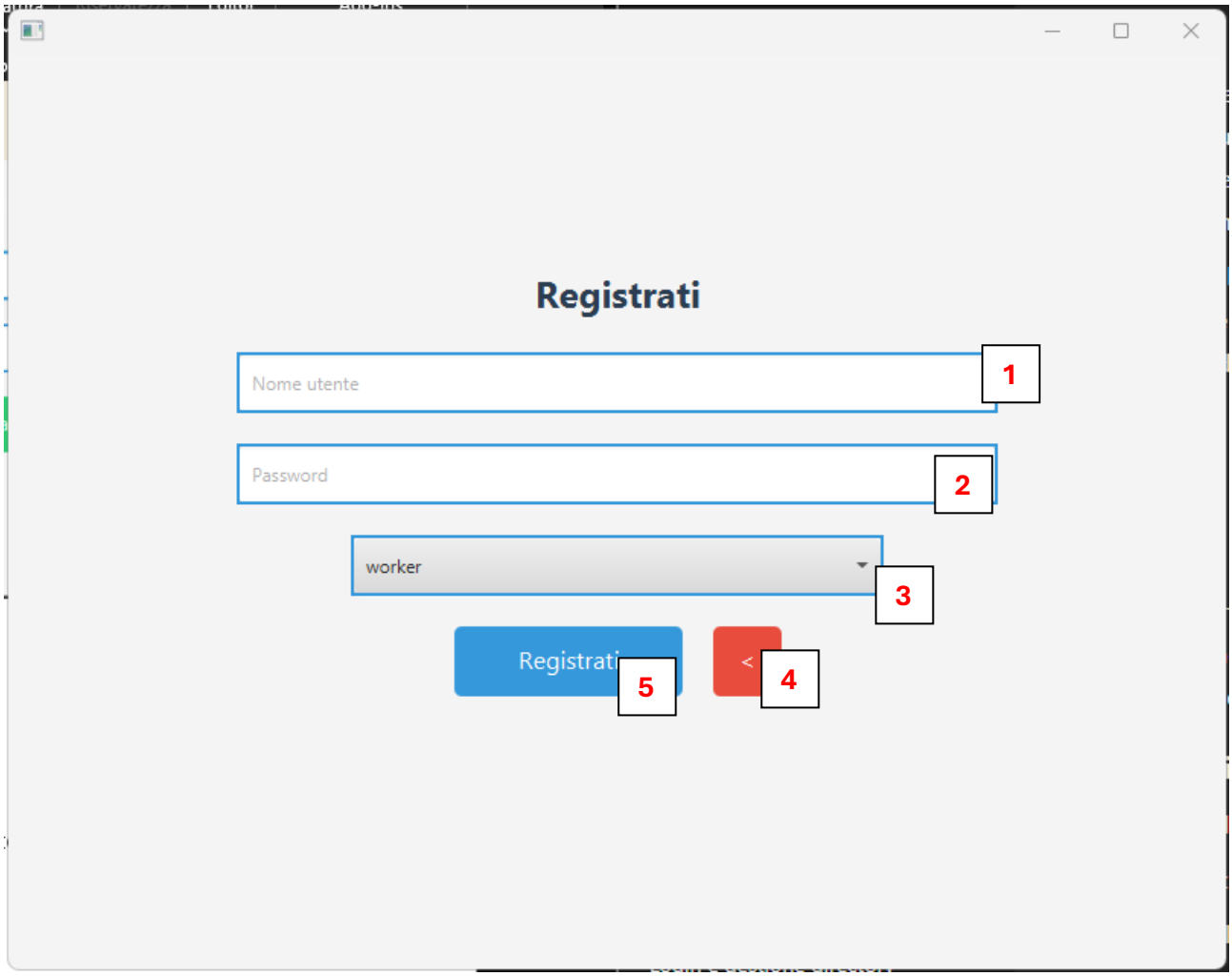
All'avvio della dell'applicazione client viene mostrata all'utente una schermata iniziale:



1. Campo di inserimento dell'username.
2. Campo di inserimento della password.
3. Bottone di accesso alla homepage.
4. Bottone di accesso alla finestra di registrazione.

In seguito alla pressione del bottone di accesso, se le credenziali sono corrette, l'utente viene automaticamente reindirizzato alla propria homepage; in caso contrario, viene mostrato in basso un messaggio di errore.

In seguito alla pressione del bottone di registrazione, l'utente viene indirizzato verso una nuova finestra:

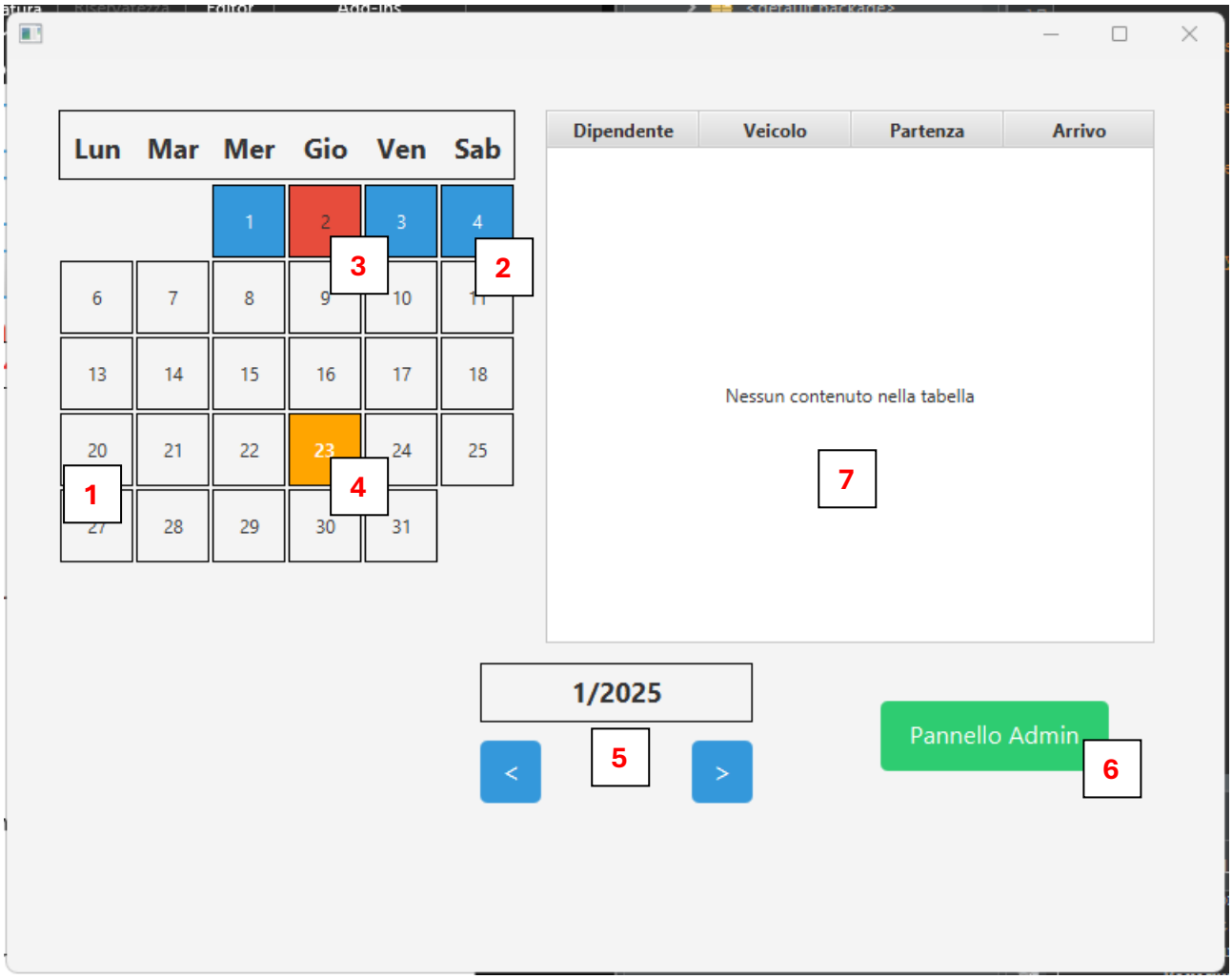


1. Campo di inserimento dell'username.
2. Campo di inserimento della password.
3. Choicebox per la selezione del ruolo dell'utente.
4. Bottone di registrazione.
5. Bottone di ritorno alla pagina precedente.

In seguito alla pressione del bottone di registrazione verrà mostrato un messaggio che esprime l'esito dell'operazione.

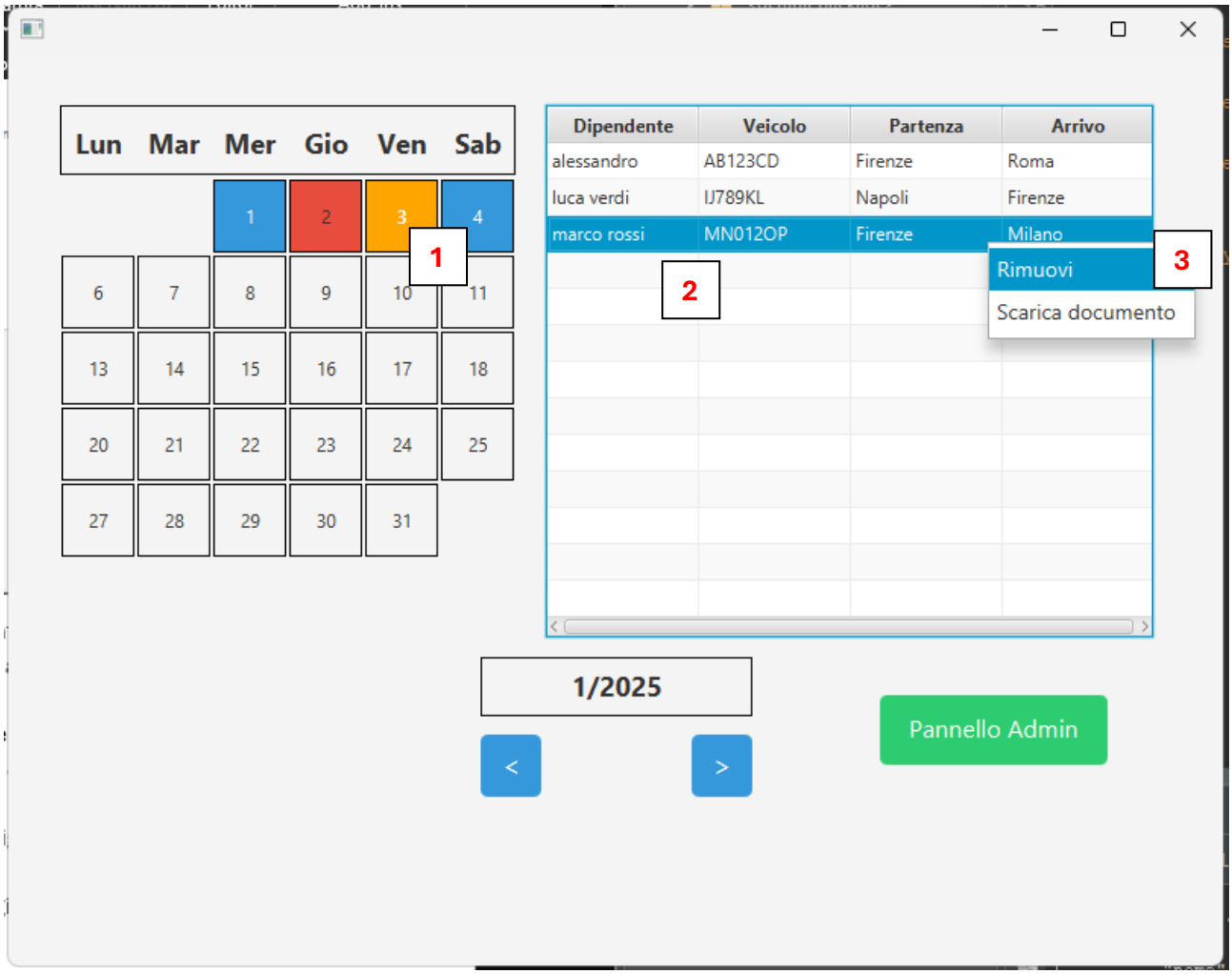
Facendo un passo indietro, se le credenziali inserite nella finestra iniziale sono corrette e viene premuto il bottone di accesso, l'utente viene indirizzato verso la sua homepage. Si tratta della finestra centrale dell'applicazione: da qui è possibile prendere visione del calendario aziendale e quindi dei viaggi previsti per il mese selezionato.

Questa finestra ha un aspetto diverso in base al ruolo dell'utente, ma, per un admin, il suo aspetto è il seguente:



1. Calendario del mese. All'avvio mostra il mese corrente.
2. I giorni in blu indicano che per quel giorno è previsto almeno un viaggio.
3. Il giorno rosso indica il giorno corrente.
4. Il giorno arancio rappresenta il giorno al momento selezionato.
5. Controlli del calendario: i due bottoni permettono di andare avanti o indietro di un mese rispetto a quello mostrato.
6. Non presente sulla GUI dei worker, il bottone di configurazione consente all'utente di essere indirizzato verso la finestra di configurazione.
7. La tabella mostra, se presenti, i viaggi previsti per il giorno selezionato.

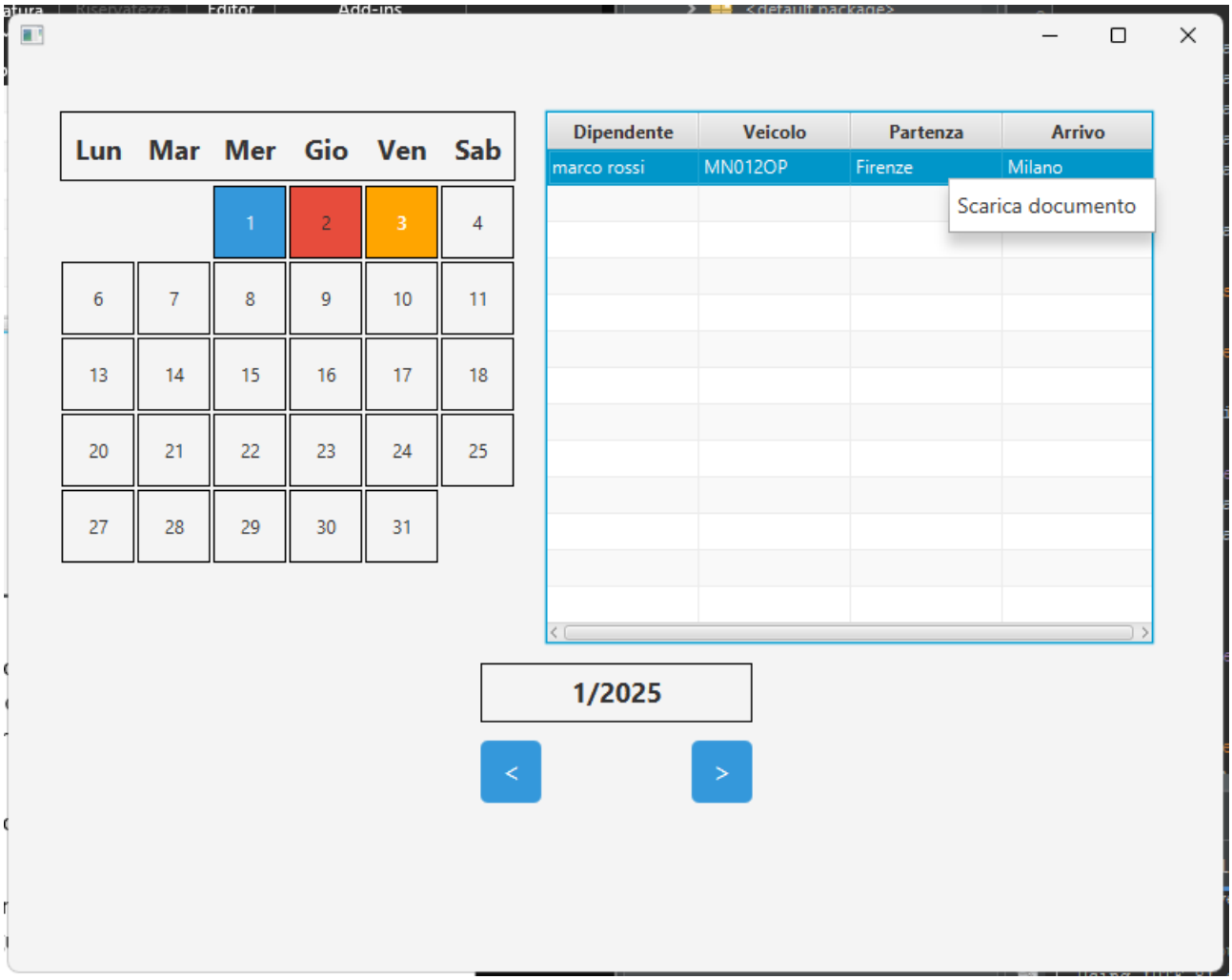
Quando il giorno selezionato contiene dei viaggi in programma, la tabella viene popolata con delle entrate:



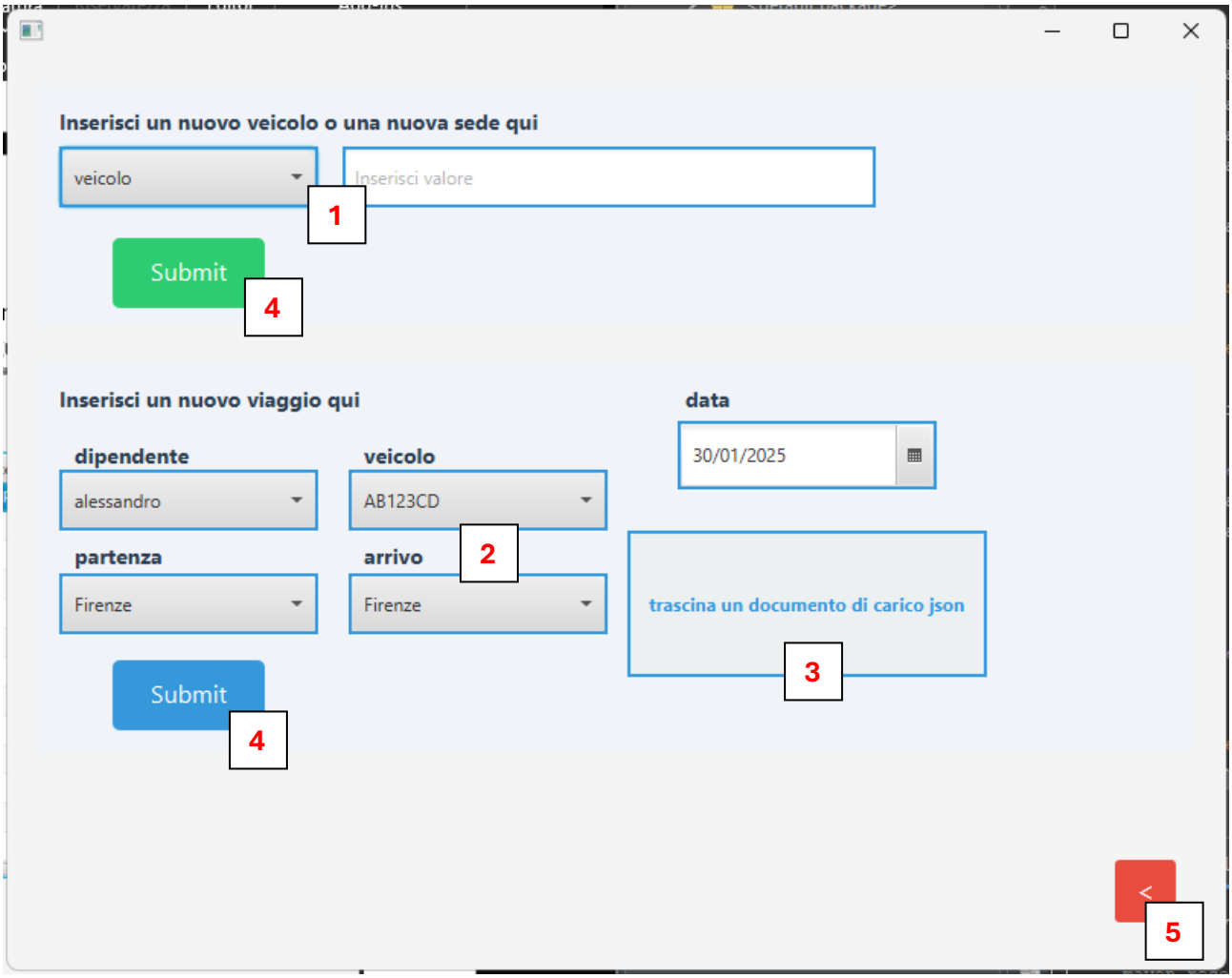
1. Il giorno selezionato, in questo caso 03-01-2025.
2. Ogni riga della tabella rappresenta un viaggio previsto per il giorno selezionato.
3. In seguito alla pressione con tasto destro di una riga della tabella viene mostrato un menù. Per i normali worker, l'opzione di "Rimuovi" è nascosta.

Come suggerisce il nome, "Rimuovi" permette ad un admin di eliminare dal database il viaggio selezionato, mentre "Scarica documento" fa in modo che il documento di carico del viaggio venga salvato in "resources/documentiDiCarico".

Per completezza, riporto di seguito anche la homepage di un nomale worker (notare gli elementi mancanti, ovvero i viaggi altrui, il bottone di configurazione e l'opzione "Rimuovi"):



L’ultima finestra implementata dall’applicativo è quella di configurazione:



1. Pannello di inserimento di nuove sedi o veicoli (l’entità di interesse può essere scelta nel vicino Choicebox.
2. Pannello di inserimento di nuovi viaggi.
3. Selettore di documenti: permette all’utente di allegare un file semplicemente trascinandolo su di esso.
4. I due bottoni di submit della nuova entrata.
5. Il bottone di ritorno alla homepage.

Unit test

Il progetto contiene uno unit test lato client.

Il test vuole mettere alla prova il meccanismo di controllo degli accessi implementato sul server. Viene simulato un utente malevolo, tale *Marco Rossi*, che, seppur worker, intende ottenere dal database tutte le informazioni messe a disposizione degli admin.

Implementazione del database

In alcuni casi le query predefinite dal framework non sono risultate sufficienti ad un efficiente funzionamento del database. Un esempio è la ricerca dei viaggi per data: anziché ottenere tutti i viaggi e poi filtrarli, ho preferito scrivere query aggiuntive utilizzando la direttiva @Query del framework.

In aggiunta a questo:

- Data la presenza di una chiave a più attributi per la tabella *Viaggio*, è stato necessario definire una classe *ViaggioId* per descriverla.
- Essendo *carico* un attributo di tipo BLOB di dimensioni anche grandi, è stata definita un'ulteriore classe, *ViaggioSenzaCarico*, che descrive un Viaggio tralasciandone però il documento di carico in modo da ridurre le dimensioni delle istanze generate.

All'avvio del server, se il database non è presente, questo viene generato a partire da uno script contenuto in "resources/initFiles/616157.sql" in maniera completamente automatica utilizzando ScriptRunner.

Utilizzo di chatGPT

Utilizzato per implementare la propagazione delle eccezioni dal server al client, per aprire le connessioni verso il server e per deserializzare a dovere una stringa Json che rappresenta un'istanza di ArrayList<DynamicMessage>.

PROMTs:

Come ritornare al client un'eccezione da framework spring.

Come posso utilizzare HttpClient in java per aprire una connessione POST verso un server?

Avendo in risposta da un server un tipo ArrayList<DynamicMessage>, dove DynamicMessage è una classe da me definita, come potrei deserializzarli sul client?