Documentazione di progetto
-
Java

POLITECNICO
MILANO 1863

Tema C

Gruppo 1
El Abiad Riad Eddin
Randazzo Lorenzo
Terraneo Andrea

1. **Struttura e Logica**

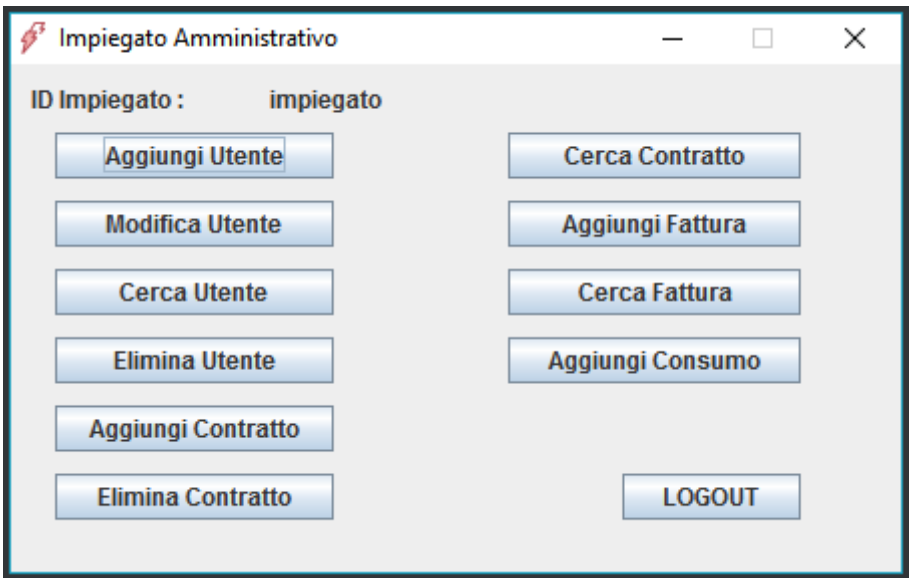
Il software permette l'accesso al sistema tramite Client RMI, ogni utente che lo utilizza può svolgere determinate funzioni a seconda del proprio privilegio.
L'implementazione dei metodi utilizzabili dagli utenti viene caricato in un registro (register binding), il client conoscendo l'indirizzo dell'host ottiene (look up) e utilizza un'interfaccia del server.
Tramite questo meccanismo di invocazione remota viene mascherata l'effettiva struttura del programma all'utente, il quale utilizza semplicemente dei metodi dell'interfaccia generale.
Questa interfaccia estende le interfacce minori relative ad ogni tipologia di utente e funzione (IntCliente, IntImpiegato, IntManutentore, IntDB). Ogni metodo delle interfacce estese è implementato in una classe dedicata, esso chiamerà un metodo del gestore del relativo oggetto che chiamerà a sua volta un metodo del DB manager, unico tramite tra programma e database.

INT GEN-->INT SPEC-->IMPL INT-->GESTORE-->DBMANAGER

Per avere accesso alle diverse funzionalità l'utente deve effetture l'accesso incrociando i suoi dati con quelli contenuti nel database, anch'esso connesso al server. Se il login va a buon fine l'utente ha accesso alla finestra dedicata. Questa finestra varia in base al privilegio dell'utente, esso può essere cliente, impiegato o manutentore. Ognuna di queste tipologie ha la possibilità di compiere le diverse azioni elencate in seguito.

Cliente
- Pagare una fattura specifica
- Cercare le proprie fatture in uno stato (Pagate o meno)
- Consultare le informazioni relative ad uno dei suoi contratti
- Consultare informazioni relative ai suoi consumi
- Segnalare un guasto relativo ad un componente
- Visualizzare i guasti relativi alla sua utenza

Impiegato
- Aggiungere un utente di qualsiasi tipo
- Eliminare un particolare utente
- Registrare le modifiche alle informazioni relative ad un utente
- Ottenere la lista di una tipologia di utente specifica
- Cercare un utente tramite il suo id, il suo numero di telefono, la sua mail
- Registrare un contratto
- Eliminare un contratto
- Cercare uno specifico contratto
- Cercare la lista dei contratti relativi ad un cliente
- Registrare una fattura
- Cercare l'elenco delle fatture relative ad un contratto
- Cercare una specifica fattura
- Mostrare l'elenco delle fatture in uno stato



Manutentore
- Registrare un guasto
- Registrare la presa in carico di un guasto
- Registrare la riparazione di un guasto
- Ottenere l'elenco dei guasti in base allo stato (Segnalato, preso in carico, risolto)
- Ottenere l'elenco dei guasti del manutentore
- Registrare un nuovo componente
- Registrare la modifica di un componente
- Eliminare un componente
- Cercare un componente specifico
- Cercare l'elenco dei componenti di un tipo
- Registrare consumi relativi ad un contratto

2. Accesso al Database

Per accedere al database, abbiamo utilizzato come interfaccia phpMyAdmin, in quanto molto comoda e intuitiva. Abbiamo provveduto, come inizio, a progettare la parte logica del database in modo tale che sia coerente con i dati che verranno poi salvati.
Sono state create 6 tabelle, in cui ognuna contiene gli attributi relativi al suo campo di competenza, più le varie chiavi esterne. Le tabelle sono state cosi create:

Utente(IDUtente, Password, Email, NumeroTelefono, Cognome, Nome, DataNascita, Indirizzo, Privilegio, Loggato)
Contratto(IDContratto, DataInizio, IDUtente, IDCompPos, IDCompNeg, Bilancio)
Componente(IDComponente, Tipo, ValoreEnergetico)
Posizione(Longitudine, Latitudine, IDComponente)
Fattura(IDFattura, DataEmissione, Importo, Dettagli, StatoFattura, IDContratto)
Guasto(IDGuasto, IDComponente, StatoGuasto, IDCliente, IDManutentore, DataSegnalazione, DataIncarico, Dettagli, DataRiparazione)

Una volta create le varie tabelle, si procede all'inserimento di almeno una riga nella tabella Utente con le varie credenziali e privilegio settato come Impiegato, in modo tale che il primo accesso è possibile eseguirlo, e da li vi sarà la possibilità di aggiungere nuovi utenti da parte dell'impiegato tramite la GUI, sia di tipo Cliente che Manutentore.

Per quanto riguarda la parte di codice in Java, la connessione, la lettura e la scrittura del DB, è gestita dalla classe DBManager di tipo singleton.
Come prima cosa bisogna rigorosamente effettuare la connessione al DB tramite il metodo connect():

```
public void connect() throws SQLException{
		this.connection = null;

		connection =
DriverManager.getConnection("jdbc:mysql://localhost:8889/db_progetto?autoreco
nnect=true&useSSL=false",
					username, password);
		System.out.println("Server connesso!");
	}
```

dove il metodo getConnection istanzia una connessione con il database, passandogli come parametri in ordine d'elenco: l'url del database, l'username e la password per accedere al database.
Dopodiché è possibile effettuare qualunque tipo di query sul DB tramite alcune semplici istruzioni.
Qui di seguito vi è un esempio di metodo che cerca un utente tramite il suo IDUtente nel DB:

```
public Utente cercaUtente(String id) throws SQLException, CercaException{

		Utente utente= new Utente();

		statement = null;
		rs = null;

		String query = "SELECT * FROM Utente WHERE IDUtente = ?";
		statement=(PreparedStatement) connection.prepareStatement(query);
		statement.setString(1, id);

		rs = statement.executeQuery();

		if(rs.next()){

			utente = new Utente(rs.getString(1), rs.getString(2),
rs.getString(3), rs.getString(4), rs.getString(5),
					rs.getString(6), rs.getDate(7), rs.getString(8),
Privilegio.valueOf(rs.getString(9)));

			rs.close();
			statement.close();
			return utente;

		}else {

			rs.close();
			statement.close();
			throw new CercaException("Utente");

		}
	}
```

Come prima cosa si crea l'oggetto di tipo PreparedStatement il quale conterrà la query che si andrà ad eseguire. Si crea la stringa della query da eseguire e la si passa al metodo prepareStatement, che ritorna un oggetto da assegnare all'oggetto creato inizialmente chiamato statement. Nella query creata nella variabile stringa, come si puo notare contiene un punto di domanda, tale punto di domanda verrà sostituito dai metodi set dell'oggetto statement, in ordine numerico: il primo punto di domanda fa riferimento all'indice 1, il secondo al 2 e così via. Una volta eseguita la query tramite il metodo executeQuery, bisogna salvare il risultato da qualche parte. Di questo se ne occupa la classe ResultSet; di conseguenza bisogna creare anche un oggetto di tipo Resulset, e salvare al suo interno il risultato che ritorna il metodo executeQuery.
Dopodichè bisogna controllare se tale query ha dato risultati o meno tramite il metodo next() che sposta il puntatore alla prima riga di tabella trovata; se la riga di tabella è piena il metodo restituisce true, senno false.
Nel caso tornasse true, come ultima cosa bisogna semplicemente salvarsi i vari campi della tabella tramite i metodi get chiamati sull'oggeto di ResultSet, passandogli come parametro un int che indica il numero della colonna della tabella dalla quale si vuole ottenere il risultato.
Infine si chiamano i metodi close() sull'oggeto di tipo PreparedStatement e ResultSet.

Nel caso in cui il metodo next() ritorna false, e dunque non si ottiene nessuna riga dalla tabella, allora viene lanciata un'eccezione che verrà poi catturata dal client.

Sono stati poi implementati tutti gli altri metodi necessari che verranno poi descritti dalla GUI.

3. Exception

Sono state create diverse classi per gestire le varie eccezioni che potrebbero occorrere durante l'eseguimento del programma:
- ListaException, eccezione che viene lanciata quando vengono chiamati i metodi dal database che devono ritornare una lista, ma non contiene nulla al suo interno, in quanto non sono stati trovati risultati nel DB.
- CercaException, eccezione che viene lanciata quando vengono chiamati i metodi dal database che devono ritornare un oggetto, o qualche valore, ma la ricerca di tale oggetto non ha avuto risultati
- LoginErratoException, eccezione che viene lanciata se il login non va a buon fine in quanto o l'utente o la password sono sbagliati
- ComponenteNonTuoException è un'eccezione che occorre nel caso si cerca di segnalare un guasto oppure nel caso si aggiungesse un contratto con un IDComponente che magari appartiene già ad un altro contratto
- GiaConnessoException vieta l'accesso ad un utente quando tale utente è già dentro!
- GiaPagataException è un'eccezione che viene lanciata quando si prova a pagare una fattura che è già stata pagata precedentemente!
- InesistenteException tale eccezione viene lanciata da diversi metodi, ed è utilizzata come controllo per verificare che prima di inserire per esempio un contratto con un determinato idutente, l'idutente effettivamente esista; se non esiste viene appunto lanciata l'eccezione.
- PrivilegiInsufficientiException è un'eccezione lanciata dal metodo prendiIncaricoGuasto, in modo da non permettere né ad un cliente né ad un impiegato di fare ciò, ma solo al manutentore.
- UtenteEsistenteException viene lanciata quando si registra un nuovo utente con un determinato idutente, ma tale id appartiene già ad un altro utente.

Queste tre eccezioni vengono lanciate dal DB e vengono catturate solo quando arrivano al client, per poi essere gestite a seconda del caso (per esempio finestra pop up di errore).

Le eccezioni SQLException, che riguardano eccezioni riguardanti richieste SQL vengono tutte catturate dal server e gestite all'interno del server.

Per quanto riguarda le NotBoundException e RemoteException anch'esse vengono gestite dal client con una finestra pop up che mostra un messaggio di errore.

4. Test JUnit

Nella sezione BeforeClass, tramite il metodo Setup(), vengono creati la connessione al db ed un utente per ciascun tipo, questi utenti vengono successivamente loggati.
Prima di ogni test vengono creati un contratto di prova, due fatture di prova, un componente e un guasto nella sezione Before, tramite il metodo preSet().
Vengono testate le seguenti funzionalità:
-test dei valori di ritorno, IDFattura e StatoFattura, degli elenchi delle fatture emesse e pagate ottenuti dall'impiegato al fine di verificare la coerenza con la lista richiesta;
-test del pagamento di una fattura da parte del cliente. Viene effettivamente verificato il cambio di stato della fattura, da emessa a pagata;
-test della presa in carico e della riparazione di un guasto da parte di un manutentore. Anche in questo test, viene controllato il cambio di stato dei guasti, da segnalato a preso in carico e da preso in carico a segnalato;
Alla fine di ognuno di questi test vengono eliminati i guasti, le fatture e il contratto di prova nella sezione After, tramite il metodo deleteSet().
Nella sezione AfterClass, tramite il metodo Confirm(), vengono prima eliminati gli utenti cliente e manutentore, successivamente viene effettuato il logout da parte dell'impiegato ed eliminato quest'ultimo dal database.

5. Grafica

Il software dispone di una grafica a finestre dedicata, nella quale da ogni finestra (frame) si può effettuare una operazione svolte anche in modi diversi (si veda il frame “ImpiegatoCercaUtente” che può ottenere un singolo utente tramite il suo IDUtente oppure una lista di utenti tramite un numero di telefono, una email oppure tramite il privilegio ad essi associato).
Ad ogni frame che debba ricevere in input dei dati oppure mostrarne è associato un messaggio d’errore (dialog) che avvisa l’utente di aver effettuato un inserimento non valido o che la ricerca sul DB non è andata a buon fine. Inoltre, se l’utente tenta di chiudere il frame principale a lui dedicato, il software mostra un dialog di conferma di uscita dal programma ed in caso di risposta affermativa effettua quindi Logout. Analogamente, se l’Utente dal frame principale clicca sul bottone Logout esso mostra lo stesso dialog. Ogni bottone ha un’azione associata, come ad esempio passare in un nuovo frame oppure fare delle operazioni sui dati (scrittura, lettura o modifica).

Nello sviluppare l’interfaccia grafica sono stati utilizzati i seguenti componenti:
-Frame come base per tutte le finestre
-Dialog come messaggi d’errore o di conferma
-Label come etichetta di testo
-TextField come linea singola editabile di testo
-ComboBox come lista selezionabile di valori
-Button come bottone pigiabile
-TextArea come area multilinea editabile di testo
-PasswordField come linea singola editabile di testo che nasconda l’inserimento
-Spinner come lista di valori limitabile superiormente e inferiormente il cui campo può essere sia editato sia incrementato/decrementato da appositi bottoni
-Table come tabella per mostrare una lista di oggetti

Dopo aver avviato il mainServer e subito dopo il mainClient, quest’ultimo lancia la prima finestra, ovvero quella di Login, dalla quale l’Utente inserisci le sue credenziali nei rispettivi IDUtente e Password e può effettuare il login. Al click del mouse sul bottone Login è associato un evento che consiste nel ricavare dai due campi di testo le credenziali e che poi esse vengano passate come parametro alla funzione login chiamata nell’interfaccia generale la quale chiama l’interfaccia del DBmanager che a sua volta chiama la sua implementazione e qui il metodo Login. Esso restituisce poi all’oggetto di tipo Login il privilegio associato all’utente oppure il sotware espone un dialog d’errore se l’utente non è presente nel DB o se le sue credenziali sono errate. Nel caso di insuccesso nel Login, compare il dialog con un messaggio d’errore. Nel

caso invece di successo, la finestra di Login, a seconda del Pivilegio associato all'utente che lo ha effettuato, apre il frame principale (ManutentoreMainFrame, ClienteMainFrame, ImpiegatoMainFrame) e chiude sé stessa. Queste finestre dispongono di molteplici bottoni che portano a nuovi frame con funzionalità differenti (ogni tipo di utente ne ha di diverse) e un bottone di Logout che funziona come spiegato precedentemente. Una volta scelta una delle funzionalità proposte e quindi raggiunto il relativo frame, si nota che il frame principale non viene chiuso, ma solo disabilitato a compiere nuove azioni fino a che non si chiuda l'ultimo frame aperto. Ciò è stato impostato per evitare che si potessero aprire diversi frame in contemporanea, anche di diverse funzionalità, o addirittura chiudere il frame principale senza portare a termine o l'operazione in corso.

6. Librerie Esterne

Nel progetto sono state utilizzate le seguenti librerie esterne:
- sql
- rmi
- util
- swing
- awt