PROTOCOLLO DELL'APPLICAZIONE

L'architettura proposta per il nostro sistema informativo è di tipo Client-Server, il server gestisce le interazioni col DBMS attraverso opportuni metodi invocati nel client. Tutti i metodi fanno riferimento a classi del package Connection.
Le richieste al server vengono gestite e formattate come "Json String" esse sono pertanto strutturate nella forma:

{action: *azione,* nick:"*nomeutente",* password:"*password",* dato:*"ValoreDato",* ….}

Tutte le richieste espresse nella forma sopradescritta possono contenere un numero indefinito di dati, dipendente dalla richiesta. Devono tuttavia necessariamente includere il parametro "action" fondamentale ai fini dell'identificazione della richiesta al server, inoltre devono essere presenti le credenziali di accesso del dipendente, al fine di evitare intrusioni nel sistema.
In aggiunta ai metodi di base necessari ad un'interazione di tipo Client-Server nel linguaggio Java (in modalità TCP sulla porta 9999), il package contiene i seguenti metodi relativi al nostro sistema:

- **cryptaPass**: riceve come argomento il nome utente del dipendente ed una password in chiaro, utilizza la password come seme e il nome utente come sale della sequenza e infine genera una password crittografata in sha 12.
  Metodo ausiliario

- **login**: riceve come argomento il nome utente e la password del dipendente, controlla che le credenziali inserite siano corrette e restituisce tutte le informazioni relative al dipendente identificato dal nome utente passato come parametro.
  Action: 0

- **visualizzaElencoCedolini**: riceve come parametro un codice fiscale e restituisce tutti i cedolini presenti nel database e relativi al dipendente identificato dal codice fiscale.
  Action: 1

- **visualizzaCedolino**: riceve come parametri il codice fiscale e l'idCedolino restituendo lo specifico cedolino identificato dall'id passato come parametro relativamente al dipendente identificato dal codice fiscale.
  Action: 2

- **timbraCartellino**: riceve come parametro un codice fiscale e assolve alla funzionalità di timbro del cartellino (ingresso e uscita) per il dipendente individuato dal codice fiscale.
  Action: 3

- **visualizzaTurniDipendente**: riceve il codice fiscale passato come argomento della funzione restituisce l'elenco dei turni relativi al dipendente identificato dal parametro.
  Action: 4

- **visualizzaElencoDipendenti**: metodo senza parametri, restituisce l'elenco dei dipendenti includendo anche tutti i dettagli.
  Action: 5

- **visualizzaDipendente**: riceve come parametro un codice fiscale e restituisce tutti i dettagli relativi al dipendente da esso individuato.
  Action: 6

- **visualizzaElencoFermate**: metodo senza parametri, restituisce l'elenco delle fermate presenti completo di tutti i dettagli
  Action: 7

- **visualizzaFermata**: riceve come parametro l'idFermata, restituisce tutti i dettagli relativi alla fermata identificata dal parametro.
  Action: 8

- **visualizzaElencoLinee**: metodo senza parametri che restituisce l'elenco di tutte le linee presenti e relativi dettagli.
  Action: 9

- **visualizzaLinea**: riceve come parametro l'idLinea, restituisce tutti i dettagli della linea conseguentemente identificata.
  Action: 10

- **visualizzaElencoAutobus**: metodo senza parametri, restituisce l'elenco degli autobus ed i relativi dettagli.
  Action: 11

- **visualizzaAutobus**: riceve come parametro la targa dell'autobus e restituisce i dettagli dell'autobus da essa individuato.
  Action: 12

- **setOrarioLinea**: riceve come parametro l'idLinea, i dettagli temporali della linea e inoltre il numero di bus consigliati, effettua l'aggiornamento della linea specifica dall'idLinea con i nuovi valori passati anch'essi come parametro.
  Action: 13

- **cambiaPassword**: riceve come parametri il codice fiscale, la vecchia password e la nuova password, consente di cambiare la password del dipendente individuato dal parametro codice fiscale, impostando la nuova password specificata come parametro, previa verifica della password corrente mandata anch'essa come parametro.
  Action: 14

- **modificaDipendente**: riceve come unico parametro una Json String dal quale ricava le informazioni da modificare e conseguentemente effettua le modifiche richieste per il dipendente specificato nel campo Json "CF" (è stato necessario utilizzare questa modalità operativa a differenza degli altri metodi poiché nell'operazione di modifica possono essere aggiornati un numero di campi non definito).
  Action: 15

- **reimpostaPassword**: riceve come parametro il codice fiscale e reimposta la password del dipendente identificato da esso ad una password generata dal sistema.
  Action: 16

- **aggiungiTurno**: riceve come parametro il codice fiscale e permette di aggiungere un turno al dipendente individuato dal parametro. E' inoltre necessario passare come argomenti anche il giorno e l'orario di inizio e fine turno.
  Action: 17

- **eliminaTurno**: riceve come parametri il codice fiscale, il giorno e l'orario d'inizio turno e rimuove il turno conseguentemente individuato.
  Action: 18

- **gestisciGuasti**: riceve come argomento la targa di un autobus e il nuovo stato, imposta il nuovo stato dell'autobus individuato attraverso la targa, viene anche passata la descrizione del guasto che può eventualmente essere nulla.
  Action: 19

- **eliminaLinea**: riceve come parametro l'idLinea ed elimina la linea conseguentemente individuata.
  Action: 20

- **eliminaFermata**: riceve come parametro l'idFermata ed elimina la fermata conseguentemente individuata.
  Action: 21

- **eliminaFermataDaLinea**: riceve come parametri idLinea e idFermata ed elimina la fermata specificata dalla linea specificata dal parametro, sostanzialmente elimina un'associazione fra linea e fermata.
  Action: 22

- **eliminaDipendente**: riceve come argomento il codice fiscale e cancella il dipendente da esso individuato.
  Action: 23

- **eliminaAutobus**: riceve come argomento la targa e cancella l'autobus da essa individuato.
  Action: 24

- **creaLinea**: riceve come argomenti tutti i parametri necessari alla creazione di una linea e la aggiunge nel database.
  Action: 25

- **creaFermate**: riceve come argomenti tutti i parametri necessari alla creazione di una fermata e la aggiunge nel database.
  Action: 26

- **compilaCedolino**: riceve come argomenti tutti i parametri necessari alla creazione di un cedolino e lo aggiunge nel database.
  Action: 27

- **associaAutistaAlinea**: riceve come parametri il codice fiscale, il giorno, l'orario d'inizio del turno e l' idLinea e assegna la linea specificata all'autista individuato dal codice fiscale nel turno individuato dagli altri parametri.
  Action: 28

- **aggiungiFermataALinea**: riceve come parametri l'idFermata e l'idLinea e aggiunge la fermata individuata dal primo parametro alla linea individuata dal secondo.
  Action: 29

- **ricercaFermateLinea**: riceve come parametro l'idLinea e restituisce l'elenco delle fermate associate alla linea individuata attraverso il parametro.
  Action: 30

- **aggiungiDipendente**: riceve come argomenti tutti i parametri necessari alla creazione di un dipendente e lo aggiunge nel database.
  Action: 31

- **aggiungiAutobus**: riceve come argomenti tutti i parametri necessari alla creazione di un autobus e lo aggiunge nel database.
  Action: 32

- **cercaFermateCittadino**: riceve come argomenti la posizione (latitudine e longitudine) ed il raggio (in metri) e restituisce tutte le fermate (e relativi dettagli) con distanza dalla posizione specificata minore del

raggio.
Action: 33

- **cercaLineaCittadino**: riceve come parametro l'idLinea e restituisce i dettagli e le fermate della linea individuata dal parametro.
  Action: 34

- **log**: scrive su un file di testo tutte le richieste pervenute al server memorizzando indirizzo IP e credenziali d'accesso del dipendente richiedente, memorizza anche data e ora della richiesta.
  Metodo ausiliario.

- **send**: permette di inviare i dati al client, se la richiesta non prevede l'invio dei dati, la funzione restituisce comunque un codice relativo allo stato di esecuzione della richiesta, 1 se l'operazione è andata a buon fine, -1 altrimenti (il metodo è in overload).
  Metodo ausiliario.

- **sendFormatError**: comunica al client con un opportuno messaggio esplicativo che la formattazione della richiesta non è corretta.
  Metodo ausiliario.

- **sendImage:** riceve come parametro il nome della foto (salvata nel server nella cartella "img"), consente di mandare al dipendente la foto individuata dal parametro.
  Action: 36

- **getRuoli:** metodo senza parametri, restituisce l'elenco dei ruoli e i relativi dettagli.
  Action: 38

- **registraUscitaAutobus:** riceve come parametro la targa e un codice fiscale, consente di registrare l'uscita dell'autobus identificato dalla targa specificata, associata all'autista identificato dal codice fiscale.
  Action: 40

- **registraEntrataAutobus:** riceve come parametro la targa, consente di registrare l'entrata dell'autobus identificato dalla targa specificata.
  Action: 41

- **pulisciFermateLinea:** riceve come parametro l'idLinea, consente di cancellare tutte le fermate associate alla linea identificata dal parametro.
  Action: 42

- **getInfoCedolino:** riceve come parametro il codice fiscale e restituisce tutti i dettagli necessari alla compilazione del cedolino, relativamente al dipendente individuato dal parametro.
  Action: 43

- **cercaFermataPiuVicina:** riceve come parametro una posizione (in latitudine e longitudine), restituisce la fermata più vicina alla posizione specificata come parametro.
  Metodo ausiliario.

- **cercaLineeFermata:** riceve come parametro l'idFermata, restituisce l'insieme delle linee passanti per la fermata specificata come parametro, con tutti i dettagli e con tutte le fermate di passaggio.
  Action: 44

- **calcolaPercorso:** riceve come parametro la posizione di partenza e di arrivo (in latitudine e longitudine) e restituisce i possibili percorsi (insieme di linee e fermate) da seguire per raggiungere il punto di arrivo specificato dalla posizione di partenza specificata. Qualora non esistesse il percorso a una linea richiama il metodo calcolaPercorsoADueVie
  Action: 45

- **calcolaPercorsoADueVie:** riceve come parametri la fermata di partenza e l'arrivo, restituisce i possibili percorsi che collegano la fermata di partenza specificata a quella di arrivo specificata tramite due linee, con una fermata intermedia di intersezione tra le linee.
  Metodo ausiliario.

- **licenziaDipendente:** riceve come parametro il codice fiscale del dipendente, consente il licenziamento del dipendente specificato dal codice fiscale inserendo un apposito riscontro nel registro assunzioni**.**
  Action: 46

- **rimuoviAssociaAutistaALinea:** riceve come parametri il codice fiscale del dipendente, il giorno del turno e l'ora di inizio e fine. Permette di rimuovere correttamente una linea dal turno di un autista**.**
  Action: 48

- **aggiungiCapolineaALinea:** riceve come parametri l'id della linea e i due capolinea, permette di definire i due capolinea della linea.
  Action: 49

- **sendCedolinoEmail:** riceve come parametri il cedolino e la email, permette di inviare il cedolino alla email specificata.
  Action: 50

- **riassumiDipendente:** riceve come parametri il codice fiscale del dipendente, permette di riassumere il dipendente desiderato.
  Action: 51

- **sendCedolinoEmail:** riceve come parametri il cedolino, permette di inviare il cedolino alla al dipendente che l'ha richiesto.
  Action: 52

- **receiveImage:** riceve come parametri il codice fiscale e l'estensione dell'immagine, consente di salvare l'immagine associata al dipendente individuato dal codice fiscale specificato nella cartella "img", attribuendole l'estensione specificata come parametro.
  Metodo ausiliario.

- **receivePdf:** riceve come parametro il nome da attribuire al file e consente di caricare un file pdf e di salvarlo nella cartella "cedolini" con il nome specificato dal parametro.
  Metodo ausiliario.

- **sendFromGMail:** riceve come parametro l'indirizzo e-mail del mittente e dei destinatari, la password del mittente, l'oggetto e il corpo del messaggio. Consente di inviare una mail dall'indirizzo del mittente specificato gli indirizzi di destinazione specificati con oggetto e testo specificati nei parametri appositi, previo controllo della password del mittente passata anch'essa come parametro.
  Metodo ausiliario.

Tutte le precedenti funzioni effettuano chiamate a metodi che risolvono operazioni di tipo generico, questo poiché le operazioni di inserimento, modifica e cancellazione sono fondamentalmente le stesse e variano solamente per i parametri passati, di seguito ne vengono forniti i prototipi.

Esecuzione selezione generica:

public List <JSONObject> selectQuery(String sql).

Esecuzione inserimento generico:

public boolean insertQuery(String table, HashMap<String, String> data1, HashMap<String, Integer> data2, HashMap<String, Double> data3).

Esecuzione aggiornamento generico:

public boolean updateQuery(String table, HashMap<String, String> data1, HashMap<String, Integer> data2, String where).

Esecuzione eliminazione generica:

public boolean deleteQuery (String table, String condition).

Gli unici tre parametri che non risultano essere autoesplicativi sono quelli segnati in rosso, si tratta di tre HashMap che consentono la mappatura delle coppie "attributo", "valore".

Al fine di concludere la discussione sul protocollo si fornisce il formato dei messaggi di risposta, anch'essi strutturati in formato "Json String":

{code: *esito,* dato:*"ValoreDato"*, ….}

Si noti che i campi dato possono eventualmente non essere presenti a seconda che la risposta del server preveda o no l'invio di dati al client.