UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e Delle Scienze di Base

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Informatica

Insegnamento di Object Orientation

Anno accademico 2023/2024

Progettazione e sviluppo del software SavingMoneyUnina

Autori:
Miriam Gaetano
N86004605

Giulia Gargiulo
N86004670

Fortunata D'Urso
N86004687

Docente:
Prof. Sergio Di Martino

GitHub: https://github.com/miriam014/Object-Orientation.git

Documentazione progetto

Fortunata D'Urso, Miriam Gaetano, Giulia Gargiulo

2024-2025

Indice

1 Descrizione del progetto

1.1 Analisi del problema

SavingMoneyUnina è un sistema progettato per la gestione delle finanze personali o familiari, che permette agli utenti di monitorare, analizzare e pianificare le proprie transazioni economiche in modo semplice, consapevole ed efficiente.
Le transazioni possono essere organizzate in specifici portafogli che permettono di raggrupparle in base alle necessità dell'utente.
Il sistema offre anche la gestione di spese ricorrenti, consentendo di programmare pagamenti periodici e di segnarli come pagati, generando automaticamente le relative transazioni. È, in più, supportata una valuta aggiuntiva (USD) con conversione automatica in Euro prima della registrazione nel database.

1.2 Gestione dell'autenticazione e accesso al sistema

Il primo passo all'interno dell'applicativo è il login, che richiede all'utente l'inserimento di nome utente e password. Questi dati vengono verificati tramite un database preesistente, garantendo un accesso sicuro e personalizzato.
Una volta effettuato il login, l'utente può accedere a tutte le funzionalità disponibili implementate dal sistema.

1.3 Gestione delle transazioni

Una delle principali funzionalità del sistema riguarda la gestione delle transazioni finanziarie. Ogni utente può registrare transazioni in entrata ed in uscita, categorizzandole automaticamente grazie alla presenza di un database che inserisce categorie predefinite. In alternativa, l'utente può scegliere manualmente la categoria di appartenenza per ciascuna transazione.
In aggiunta, il sistema permette la visualizzazione delle transazioni fornendo le rispettive categorie di appartenenza, le carte di credito utilizzate e periodi temporali definiti. In questo modo, l'utente può facilmente monitorare e analizzare le proprie spese ed entrate in base alle proprie necessità.

1.4 Gestione dei portafogli

Un'ulteriore funzionalità importante è quella che riguarda l'utilizzo di portafogli. Questi permettono di raggruppare le transazioni in base alle necessità dell'utente. I portafogli posson essere personali o familiari, ovvero che permettono di raggrupare determinate transazioni appartenenti a membri di una specifica famiglia di cui l'utente fa parte.
L'utente riesce visualizzare, per ogni portafoglio, il suo saldo e tutti i dettagli delle transazioni che ne fanno parte.
L'utente può, inoltre, decidere di aggiungere manualmente una transazione in uno specifico portafoglio già esistente e creare o eliminare un portafoglio in qualsiasi momento.

1.5 Report statistici

Una funzionalità avanzata del sistema è la possibilità di generare report statistici mensili per ciascuna carta di credito. Selezionando un mese di riferimento, l'utente può visualizzare:
– La massima, minima e media delle entrate e delle uscite;
– Il saldo iniziale e finale della carta nel periodo selezionato.

Queste informazioni permettono un'analisi approfondita dello stato finanziario dell'utente per ogni singola carta, contribuendo a una gestione più consapevole delle proprie risorse.

1.6 Gestione delle spese ricorrenti

Un'altra caratteristica importante del sistema è la gestione delle spese ricorrenti.
Gli utenti possono inserire e programmare pagamenti automatici per voci come bollette mensili, rate di prestiti o altre spese periodiche. Ogni spesa ricorrente è associata a una carta di credito e viene automaticamente contabilizzata nelle categorie predefinite.
Il sistema provvede anche ad aggiornare periodicamente l'elenco delle spese ricorrenti, segnalando le scadenze e permettendo all'utente di marcare come "pagata" una specifica scadenza, registrando automaticamente una transazione di uscita.
Questo semplifica enormemente la gestione finanziaria, permettendo all'utente di rimanere sempre aggiornato sugli impegni economici futuri.

1.7 Gestione delle famiglie

Il sistema permette a un utente di appartenere a più nuclei familiari, ciascuno con il proprio gruppo di membri e transazioni associate. Per ogni famiglia, l'utente può:
– Visualizzare tutte le transazioni effettuate dai membri della famiglia, ottenendo una panoramica completa delle spese e delle entrate condivise.
– Selezionare un singolo membro della famiglia per filtrare e visualizzare esclusivamente le transazioni di quella persona.

Questa funzionalità offre una gestione finanziaria dettagliata e flessibile, consentendo agli utenti di monitorare sia l'andamento economico complessivo della famiglia, sia le singole attività finanziarie dei suoi membri.

1.8 Multi-valuta

Infine, per soddisfare le esigenze di utenti che effettuano transazioni in dollaro statunitense (USD), oltre alla valuta predefinita, l'euro (EUR), il sistema applica automaticamente un tasso di cambio predefinito (1 EUR = 1.06 USD) per convertire le transazioni inserite in USD in euro.

2 Struttura del progetto

Il sistema è sviluppato utilizzando Java con interfaccia grafica tramite JavaFX, offrendo un'esperienza utente intuitiva e completa.

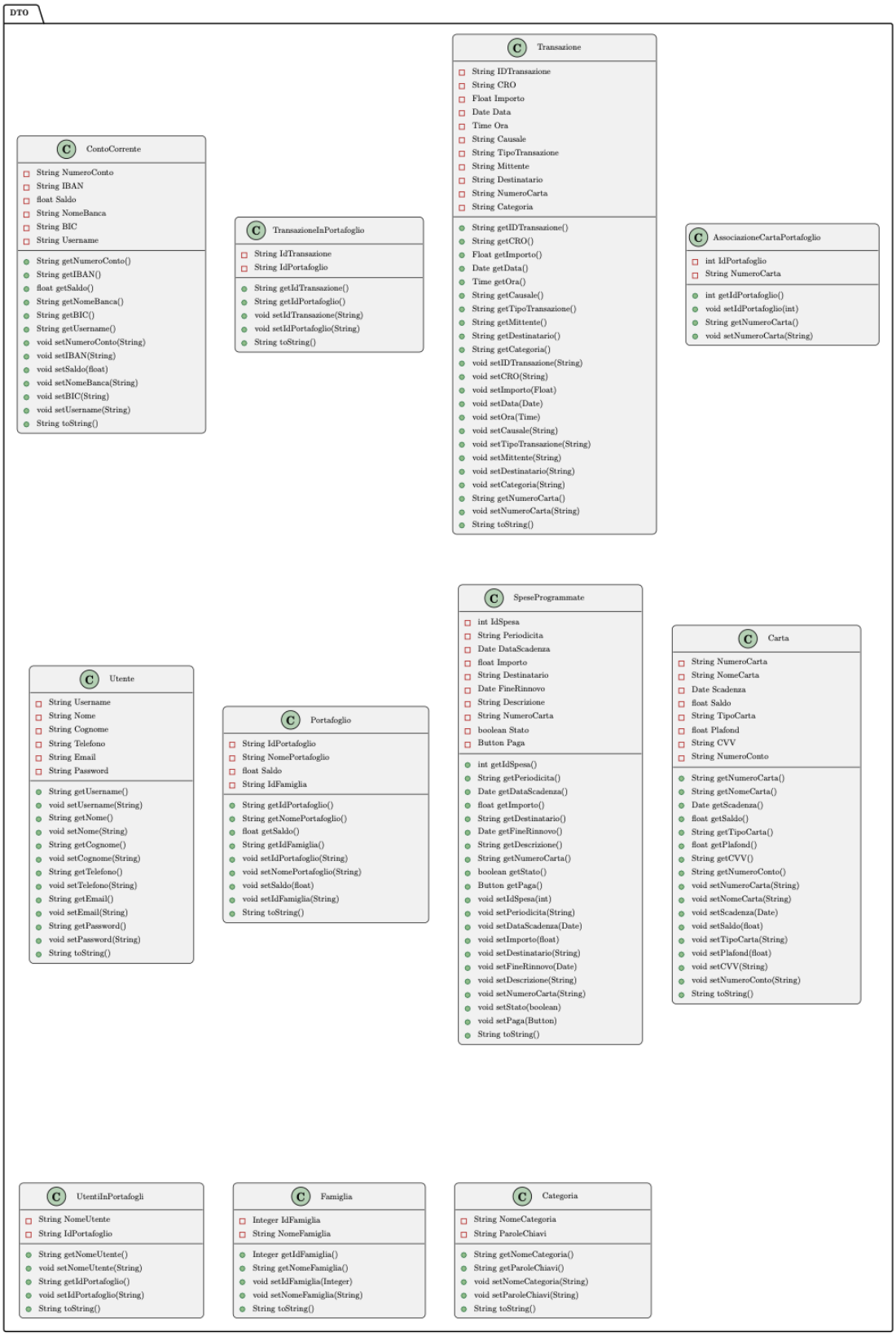
Ecco come è strutturato:

Componente	Descrizione
Controller	Gestiscono la logica dell'interfaccia utente, interagendo con le viste JavaFX e coordinando le operazioni tra la GUI e il livello logico dell'applicazione. Ricevono input dall'utente e invocano i metodi appropriati nei servizi o DAO.
DAO	Definisce le interfacce per l'accesso ai dati. Ogni DAO rappresenta un'entità del database e fornisce metodi per eseguire operazioni CRUD (Create, Read, Update, Delete).
DAO_Implementation	Implementa le interfacce definite nei DAO. Qui vengono eseguite le query SQL per interagire con il database tramite la comunicazione diretta con il database.
DTO	Contiene le classi che rappresentano i dati scambiati tra i vari livelli dell'applicazione. Servono per trasferire informazioni in modo strutturato tra il database, il controller e la UI.
Database	Classe che gestisce la connessione al database PostgreSQL. Implementa un Singleton, garantendo un'unica connessione attiva per tutta l'applicazione.
Main	Punto di ingresso dell'applicazione JavaFX. Avvia la finestra principale e imposta la prima schermata (`login.fxml`). Controlla la modalità schermo intero e memorizza le dimensioni della finestra per mantenere la stessa esperienza utente tra le schermate.
Sessione	Classe Singleton che mantiene i dati dell'utente autenticato durante l'esecuzione dell'applicazione. Contiene informazioni come l'utente loggato, le carte associate, i portafogli e le famiglie. Si interfaccia con i DAO per caricare e aggiornare i dati dell'utente senza doverli recuperare ogni volta dal database.

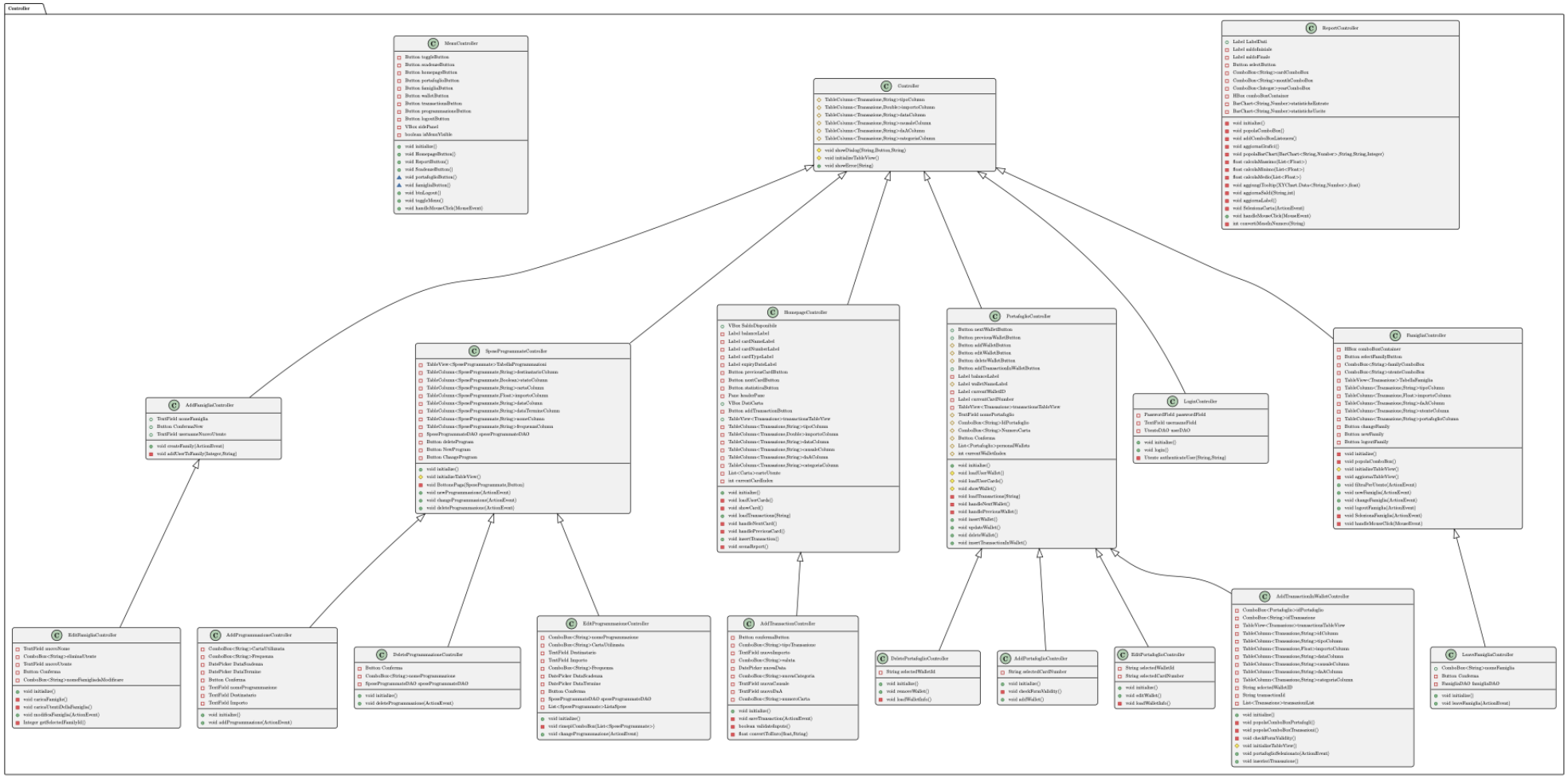
3 Class Diagram

3.1 DTO

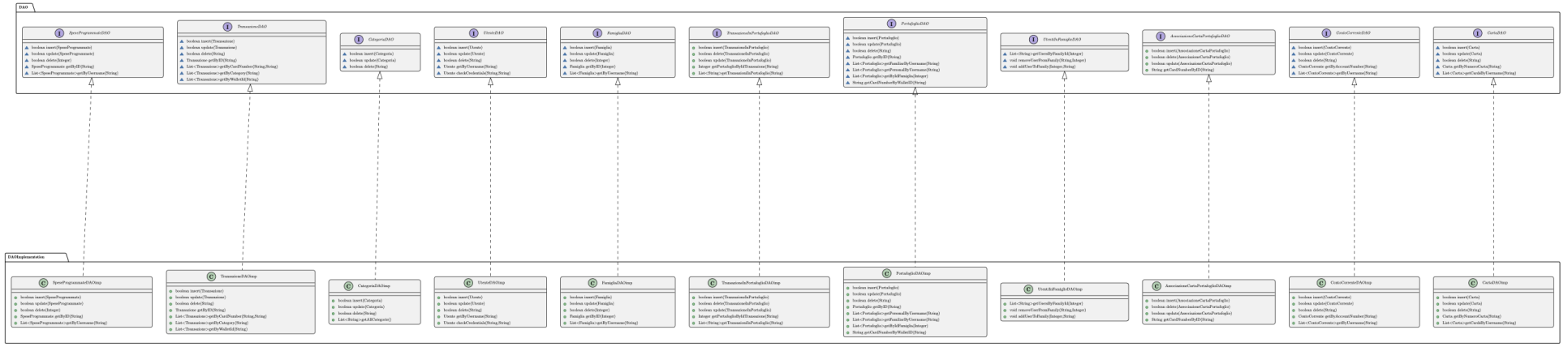
3.2 DTO


DTO
ContoCorrente
String NumeroConto
String IBAN
float Saldo
String NomeBanca
String BIC
String Username
String getNumeroConto()
String getIBAN()
float getSaldo()
String getNomeBanca()
String getBIC()
String getUsername()
void setNumeroConto(String)
void setIBAN(String)
void setSaldo(float)
void setNomeBanca(String)
void setBIC(String)
void setUsername(String)
String toString()
TransazioneInPortafoglio
String IdTransazione
String IdPortafoglio
String getIdTransazione()
String getIdPortafoglio()
void setIdTransazione(String)
void setIdPortafoglio(String)
String toString()
Transazione
String IDTransazione
String CRO
Float Importo
Date Data
Time Ora
String Causale
String TipoTransazione
String Mittente
String Destinatario
String NumeroCarta
String Categoria
String getIDTransazione()
String getCRO()
Float getImporto()
Date getData()
Time getOra()
String getCausale()
String getTipoTransazione()
String getMittente()
String getDestinatario()
String getCategoria()
void setIDTransazione(String)
void setCRO(String)
void setImporto(Float)
void setData(Date)
void setOra(Time)
void setCausale(String)
void setTipoTransazione(String)
void setMittente(String)
void setDestinatario(String)
void setCategoria(String)
String getNumeroCarta()
void setNumeroCarta(String)
String toString()
AssociazioneCartaPortafoglio
int IdPortafoglio
String NumeroCarta
int getIdPortafoglio()
void setIdPortafoglio(int)
String getNumeroCarta()
void setNumeroCarta(String)
Utente
String Username
String Nome
String Cognome
String Telefono
String Email
String Password
String getUsername()
void setUsername(String)
String getNome()
void setNome(String)
String getCognome()
void setCognome(String)
String getTelefono()
void setTelefono(String)
String getEmail()
void setEmail(String)
String getPassword()
void setPassword(String)
String toString()
Portafoglio
String IdPortafoglio
String NomePortafoglio
float Saldo
String IdFamiglia
String getIdPortafoglio()
String getNomePortafoglio()
float getSaldo()
String getIdFamiglia()
void setIdPortafoglio(String)
void setNomePortafoglio(String)
void setSaldo(float)
void setIdFamiglia(String)
String toString()
SpeseProgrammate
int IdSpesa
String Periodicita
Date DataScadenza
float Importo
String Destinatario
Date FineRinnovo
String Descrizione
String NumeroCarta
boolean Stato
Button Paga
int getIdSpesa()
String getPeriodicita()
Date getDataScadenza()
float getImporto()
String getDestinatario()
Date getFineRinnovo()
String getDescrizione()
String getNumeroCarta()
boolean getStato()
Button getPaga()
void setIdSpesa(int)
void setPeriodicita(String)
void setDataScadenza(Date)
void setImporto(float)
void setDestinatario(String)
void setFineRinnovo(Date)
void setDescrizione(String)
void setNumeroCarta(String)
void setStato(boolean)
void setPaga(Button)
String toString()
Carta
String NumeroCarta
String NomeCarta
Date Scadenza
float Saldo
String TipoCarta
float Plafond
String CVV
String NumeroConto
String getNumeroCarta()
String getNomeCarta()
Date getScadenza()
float getSaldo()
String getTipoCarta()
float getPlafond()
String getCVV()
String getNumeroConto()
void setNumeroCarta(String)
void setNomeCarta(String)
void setScadenza(Date)
void setSaldo(float)
void setTipoCarta(String)
void setPlafond(float)
void setCVV(String)
void setNumeroConto(String)
String toString()
UtentiInPortafogli
String NomeUtente
String IdPortafoglio
String getNomeUtente()
void setNomeUtente(String)
String getIdPortafoglio()
void setIdPortafoglio(String)
String toString()
Famiglia
Integer IdFamiglia
String NomeFamiglia
Integer getIdFamiglia()
String getNomeFamiglia()
void setIdFamiglia(Integer)
void setNomeFamiglia(String)
String toString()
Categoria
String NomeCategoria
String ParoleChiavi
String getNomeCategoria()
String getParoleChiavi()
void setNomeCategoria(String)
void setParoleChiavi(String)
String toString()

3.3 Controller



3.4 DAO e DAO_Implementation

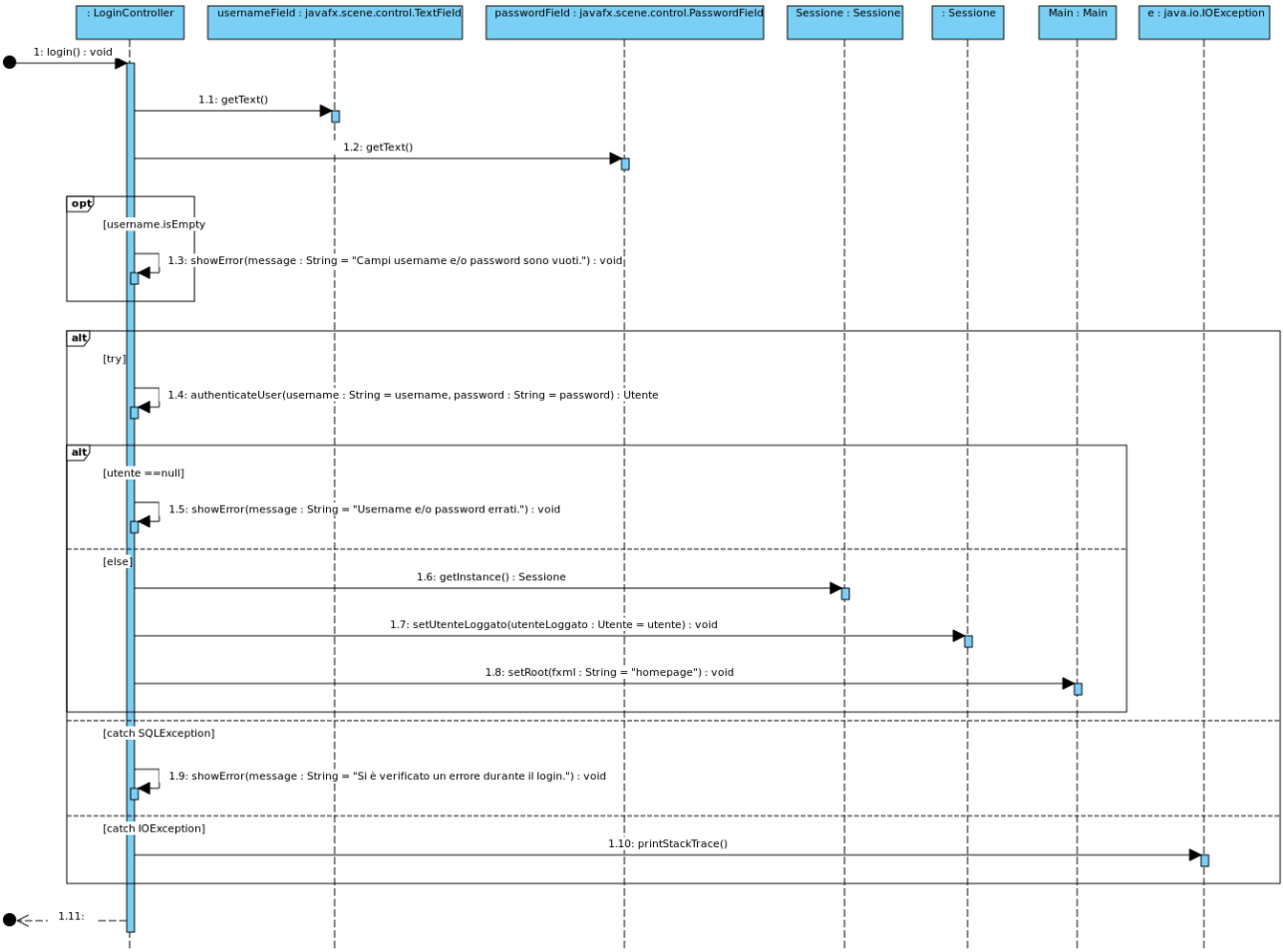


4 Sequence Diagrams

L'idea alla base del sequence diagram è rappresentare visivamente il flusso delle operazioni all'interno delle classi, evidenziando come queste si evolvono nel tempo in risposta agli input dell'utente e alle risposte di altre entità.
Questi diagrammi sono particolarmente utili per analizzare processi complessi.
Ogni azione è associata a una chiamata o a un metodo specifico e può includere logiche condizionali, come gli if, e cicli, come i loop.

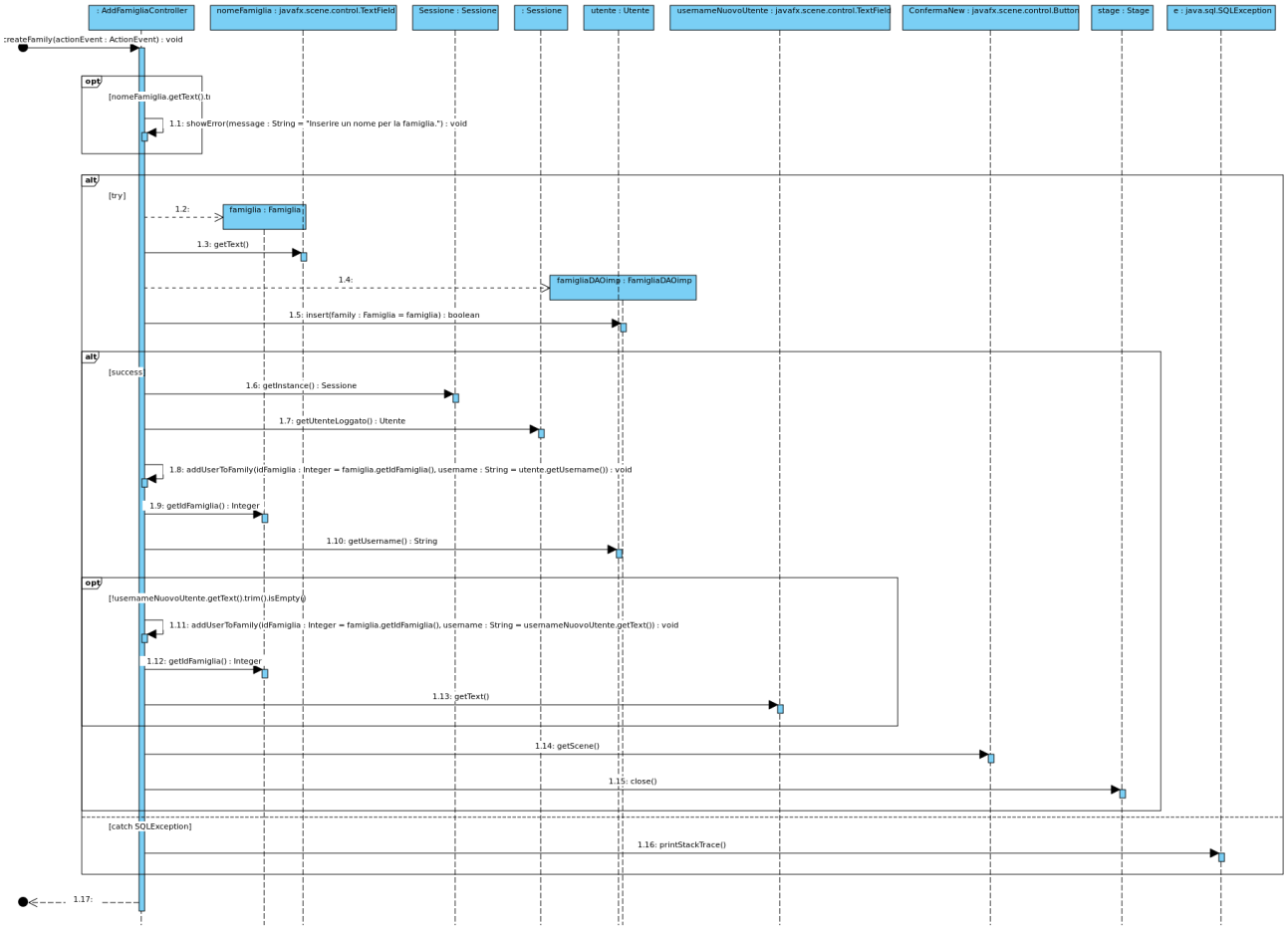
Di seguito abbiamo deciso di creare due esempi di sequence diagram.

4.1 Funzionalità "Login"



Il processo di login inizia con la raccolta del nome utente e della password dai campi di input dell'interfaccia grafica. Il sistema verifica se i campi sono vuoti e, in tal caso, mostra un messaggio di errore.
Se i dati sono presenti, viene chiamato il metodo authenticateUser(), che controlla le credenziali nel database tramite il DAO. Se l'utente non esiste, viene notificato l'errore; altrimenti, l'utente viene salvato nella sessione (Sessione.getInstance().setUtenteLoggato(utente)) e l'interfaccia viene aggiornata per mostrare la homepage.
Il sistema gestisce anche eventuali errori tecnici, mostrando un messaggio in caso di problemi con il database (SQLException) o stampando lo stack trace per errori generici (IOException).

4.2 Funzionalità ”Aggiungi Nuova Famiglia”



Quando l’utente vuole creare una nuova famiglia, inserisce il nome della famiglia nel campo nomeFamiglia e, opzionalmente, un utente inserendo il suo username nel campo usernameNuovoUtente. Preme il pulsante ConfermaNew, che attiva il metodo createFamily(ActionEvent actionEvent). Questa controlla se il campo nomeFamiglia è vuoto. Se lo è, viene mostrato un messaggio di errore con showError(”Inserire un nome per la famiglia.”) e l’esecuzione viene interrotta. In caso contrario, il sistema procede creando un oggetto Famiglia con il nome inserito e prova a salvarlo nel database usando famigliaDAOimp.insert(famiglia). Se l’inserimento non va a buon fine, viene stampato un messaggio di errore. Se invece l’operazione ha successo, si passa alla fase successiva.
A questo punto, l’utente loggato viene recuperato dalla sessione con Sessione.getInstance().getUtenteLoggato(), e il suo username viene usato per aggiungerlo alla nuova famiglia tramite il metodo addUserToFamily(famiglia.getIdFamiglia(), utente.getUsername()).
Se l’utente ha inserito un secondo username nel campo usernameNuovoUtente, anche questo viene aggiunto alla famiglia chiamando di nuovo addUserToFamily(famiglia.getIdFamiglia(), usernameNuovoUtente.getText()).
Una volta completata la creazione e l’associazione degli utenti alla famiglia, la finestra viene chiusa utilizzando stage.close(). Infine, se durante il processo si verifica un problema con il database, viene catturata un’eccezione SQLException, e lo stack trace viene stampato per il debug