Progetto di Tecnologie Informatiche per il Web

Negri Riccardo

18 giugno 2022

POLITECNICO
MILANO 1863

Docente: Fraternali Piero
Studente: Negri Riccardo 10729927 936820

Indice

1 Specifiche

1.1 Versione pure HTML

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password". La registrazione controlla l'unicità dello username. Un utente ha un nome, un cognome, uno username e uno o più conti correnti. Un conto ha un codice, un saldo, e i trasferimenti fatti (in uscita) e ricevuti (in ingresso) dal conto. Un trasferimento ha una data, un importo, un conto di origine e un conto di destinazione. Quando l'utente accede all'applicazione appare una pagina LOGIN per la verifica delle credenziali. In seguito all'autenticazione dell'utente appare l'HOME page che mostra l'elenco dei suoi conti. Quando l'utente seleziona un conto, appare una pagina STATO DEL CONTO che mostra i dettagli del conto e la lista dei movimenti in entrata e in uscita, ordinati per data discendente. La pagina contiene anche una form per ordinare un trasferimento. La form contiene i campi: codice utente destinatario, codice conto destinatario, causale e importo. All'invio della form con il bottone INVIA, l'applicazione controlla che il conto di destinazione appartenga all'utente specificato e che il conto origine abbia un saldo superiore o uguale all'importo del trasferimento. In caso di mancanza di anche solo una condizione, l'applicazione mostra una pagina con un avviso di fallimento che spiega il motivo del mancato trasferimento. Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati dell'importo trasferito e i dati del conto di origine e di destinazione con i rispettivi saldi precedenti al trasferimento e aggiornati dopo il trasferimento. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato, il conto di origine deve essere accreditato. Ogni pagina contiene un collegamento per tornare alla pagina precedente. L'applicazione consente il logout dell'utente.

1.2 Versione RIA

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password", anche a lato client.
- Dopo il login, l'intera applicazione è realizzata con un'unica pagina.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.
- I controlli di validità dei dati di input (ad esempio importo non nullo e maggiore di zero) devono essere realizzati anche a lato client.
- L'avviso di fallimento è realizzato mediante un messaggio nella pagina che ospita l'applicazione.
- L'applicazione chiede all'utente se vuole inserire nella propria rubrica i dati del destinatario di un trasferimento andato a buon fine non ancora presente. Se l'utente conferma, i dati sono memorizzati nella base di dati e usati per semplificare l'inserimento. Quando l'utente crea un trasferimento, l'applicazione propone mediante una funzione di auto-completamento i destinatari in rubrica il cui codice corrisponde alle lettere inserite nel campo codice utente destinatario.

2 Database design

2.1 Analisi testo delle specifiche

Legenda: entità, attributi, relazioni.

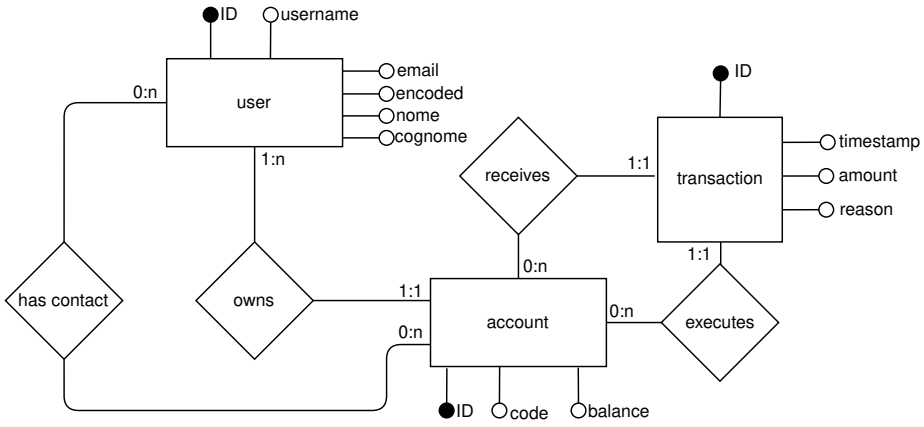
Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password". La registrazione controlla l'unicità dello username. Un utente ha un nome, un cognome, uno username e uno o più conti correnti. Un conto ha un codice, un saldo, e i trasferimenti fatti (in uscita) e ricevuti (in ingresso) dal conto. Un trasferimento ha una data, un importo, un conto di origine e un conto di destinazione. Quando l'utente accede all'applicazione appare una pagina LOGIN per la verifica delle credenziali. In seguito all'autenticazione dell'utente appare l'HOME page che mostra l'elenco dei suoi conti. Quando l'utente seleziona un conto, appare una pagina STATO DEL CONTO che mostra i dettagli del conto e la lista dei movimenti in entrata e in uscita, ordinati per data discendente. La pagina contiene anche una form per ordinare un trasferimento. La form contiene i campi: codice utente destinatario, codice conto destinatario, causale e importo. All'invio della form con il bottone INVIA, l'applicazione controlla che il conto di destinazione appartenga all'utente specificato e che il conto origine abbia un saldo superiore o uguale all'importo del trasferimento. In caso di mancanza di anche solo una condizione, l'applicazione mostra una pagina con un avviso di fallimento che spiega il motivo del mancato trasferimento. Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati dell'importo trasferito e i dati del conto di origine e di destinazione con i rispettivi saldi precedenti al trasferimento e aggiornati dopo il trasferimento. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato, il conto di origine deve essere accreditato. Ogni pagina contiene un collegamento per tornare alla pagina precedente. L'applicazione consente il logout dell'utente.

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password", anche a lato client.
- Dopo il login, l'intera applicazione è realizzata con un'unica pagina.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.

- I controlli di validità dei dati di input (ad esempio importo non nullo e maggiore di zero) devono essere realizzati anche a lato client.
- L'avviso di fallimento è realizzato mediante un messaggio nella pagina che ospita l'applicazione.
- L'applicazione chiede all'utente se vuole inserire nella propria rubrica i dati del destinatario di un trasferimento andato a buon fine non ancora presente. Se l'utente conferma, i dati sono memorizzati nella base di dati e usati per semplificare l'inserimento. Quando l'utente crea un trasferimento, l'applicazione propone mediante una funzione di auto-completamento i destinatari in rubrica il cui codice corrisponde alle lettere inserite nel campo codice utente destinatario.

2.2 Progetto concettuale - Diagramma E-R (Entità-Relazione)



2.3 Progetto logico

USER(ID, username, email, encoded, name, surname)
ACCOUNT(ID, code, balance, user)
TRANSACTION(ID, timestamp, amount, reason, origin, destination)
CONTACT(owner, element)

2.4 Database schema

Nel db tutti gli on delete sono no action perché ho la necessità di mantenere le informazioni per il futuro (da argomentare meglio)
Statement per creare il database:

```
CREATE DATABASE tiw_db;
```

Statements per creare le tabelle:

```
CREATE TABLE `tiw_db`.`user` (
`id` INT NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL UNIQUE,
`email` VARCHAR(45) NOT NULL UNIQUE,
`encoded` VARCHAR(255) NOT NULL,
`name` VARCHAR(45) NOT NULL,
`surname` VARCHAR(45) NOT NULL,
PRIMARY KEY (`id`)
);

CREATE TABLE `tiw_db`.`account` (
`id` INT NOT NULL AUTO_INCREMENT,
`code` CHAR(12) NOT NULL UNIQUE,
`balance` FLOAT NOT NULL DEFAULT 0,
`user` INT NOT NULL,
PRIMARY KEY (`id`),
CONSTRAINT `user_account` FOREIGN KEY (`user`)
REFERENCES `tiw_db`.`user` (`id`)
ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE `tiw_db`.`transaction` (
`id` INT NOT NULL AUTO_INCREMENT,
`timestamp` DATETIME DEFAULT CURRENT_TIMESTAMP,
`amount` FLOAT NOT NULL,
`reason` VARCHAR(255) NOT NULL,
`origin` INT NOT NULL,
`destination` INT NOT NULL,
PRIMARY KEY (`id`),
CONSTRAINT `transaction_origin` FOREIGN KEY (`origin`)
REFERENCES `tiw_db`.`account` (`id`)
ON DELETE NO ACTION ON UPDATE CASCADE,
CONSTRAINT `transaction_destination` FOREIGN KEY (`destination`)
REFERENCES `tiw_db`.`account` (`id`)
ON DELETE NO ACTION ON UPDATE CASCADE
);
```

```
CREATE TABLE `tiw_db`.`contact` (
`owner` INT,
`element` INT,
PRIMARY KEY (`owner`, `element`),
CONSTRAINT `contact_owner` FOREIGN KEY (`owner`)
REFERENCES `tiw_db`.`user` (`id`)
ON DELETE NO ACTION ON UPDATE CASCADE,
CONSTRAINT `contact_element` FOREIGN KEY (`element`)
REFERENCES `tiw_db`.`account` (`id`)
ON DELETE NO ACTION ON UPDATE CASCADE
);
```

2.5 Hashing della password

Si è deciso di usare Argon2 come algoritmo di hash delle password. Argon2 applica una funzione pseudocasuale alla password o passphrase di input insieme a un valore `salt` e ripete il processo molte volte per produrre una chiave derivata, che può quindi essere utilizzata come chiave crittografica nelle operazioni successive. Il lavoro di calcolo aggiunto rende molto più difficile il cracking delle password ed è noto come `key stretching`. Infatti è volutamente dispendioso in termini computazionali.

Il motivo per cui si utilizza un `salt` durante l'hashing delle password è che randomizzano l'hash generato per lo stesso valore di input. Quindi, se due utenti hanno la stessa password, i loro hash saranno comunque diversi. Ciò rallenterà ulteriormente il processo di decifrazione della password da parte di un avversario.
Argon2 genera internamente un nuovo `salt` per ogni nuova richiesta di generare un hash. Quindi, se si richiede un hash per lo stesso testo in chiaro più volte, ogni volta si ottiene un hash diverso.

Di seguito un esempio di come viene salvata nel database l'informazione necessaria per verificare la password di un utente:
`$argon2i$v=19$m=65536,t=10,p=1$PCMn0NBsYymo2jJIeoVwsA$rmrwpTVQks5s LsUKBfImzV9/b0HwF9hZKe1b5dQWwYo`
Analisi dei valori:

- `argon2i` indica la variante di Argon2 in uso;
- `v=19` indica la versione di Argon;
- `m=65536` indica l'utilizzo di memoria;
- `t=10` indica le iterazioni;
- `p=1` indica il numero di thread da utilizzare;
- `PCMn0NBsYymo2jJIeoVwsA` è il `salt` di 16 byte;

- `rmrwpTVQks5sLsUKBfImzV9/b0HwF9hZKe1b5dQWwYo` è l'`hash` di 32 byte.

Sul computer in uso al momento la validazione di una password rispetto all'`encoded` sopra ripotato impiega circa 400ms.

2.6 Script Python per popolare il database

Al fine di popolare automaticamente un nuovo database è stato creato uno script in Python chiamato "populate-tiw-db.py" .

3 Versione pure HTML

3.1 Analisi requisiti delle specifiche

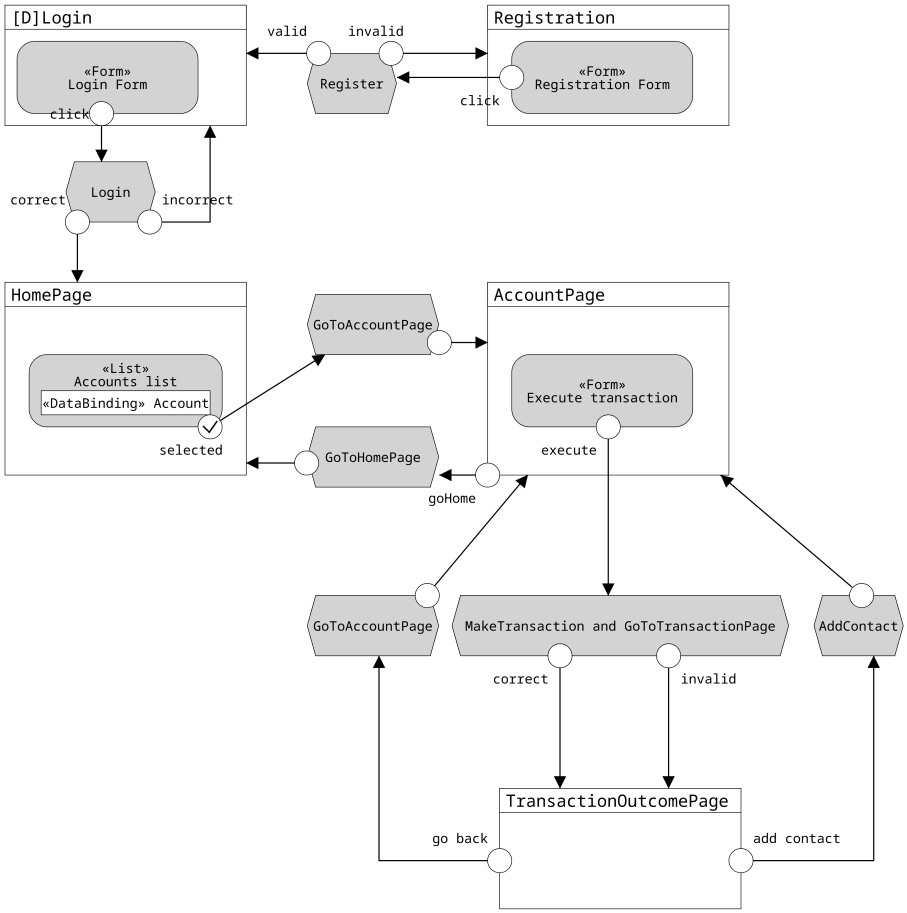
Legenda: pagine, view components, eventi, azioni (modifca il db).

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password". La registrazione controlla l'unicità dello username. Un utente ha un nome, un cognome, uno username e uno o più conti correnti. Un conto ha un codice, un saldo, e i trasferimenti fatti (in uscita) e ricevuti (in ingresso) dal conto. Un trasferimento ha una data, un importo, un conto di origine e un conto di destinazione. Quando l'utente accede all'applicazione appare una pagina LOGIN per la verifica delle credenziali. In seguito all'autenticazione dell'utente appare l'HOME page che mostra l'elenco dei suoi conti. Quando l'utente seleziona un conto, appare una pagina STATO DEL CONTO che mostra i dettagli del conto e la lista dei movimenti in entrata e in uscita, ordinati per data discendente. La pagina contiene anche una form per ordinare un trasferimento. La form contiene i campi: codice utente destinatario, codice conto destinatario, causale e importo. All'invio della form con il bottone INVIA, l'applicazione controlla che il conto di destinazione appartenga all'utente specificato e che il conto origine abbia un saldo superiore o uguale all'importo del trasferimento. In caso di mancanza di anche solo una condizione, l'applicazione mostra una pagina con un avviso di fallimento che spiega il motivo del mancato trasferimento. Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati dell'importo trasferito e i dati del conto di origine e di destinazione con i rispettivi saldi precedenti al trasferimento e aggiornati dopo il trasferimento. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato, il conto di origine deve essere accreditato. Ogni pagina contiene un collegamento per tornare alla pagina precedente. L'applicazione consente il logout dell'utente.

3.2 Completamento specifiche

- E' stata creata anche una pagina per visualizzare le informazioni dell'utente
- Si è deciso di implementare la gestione dei contatti e la possibilità di aggiungere un conto ai contatti anche nella versione pure HTML
- Nella rubrica viene aggiunto solo un conto relativo ad un utente, non tutti i conti di un singolo utente
- La rubrica è relativa all'utente e non al conto, quindi tutte le pagine dei conti di un utente vedranno la stessa rubrica

3.3 Application Design



3.4 Componenti

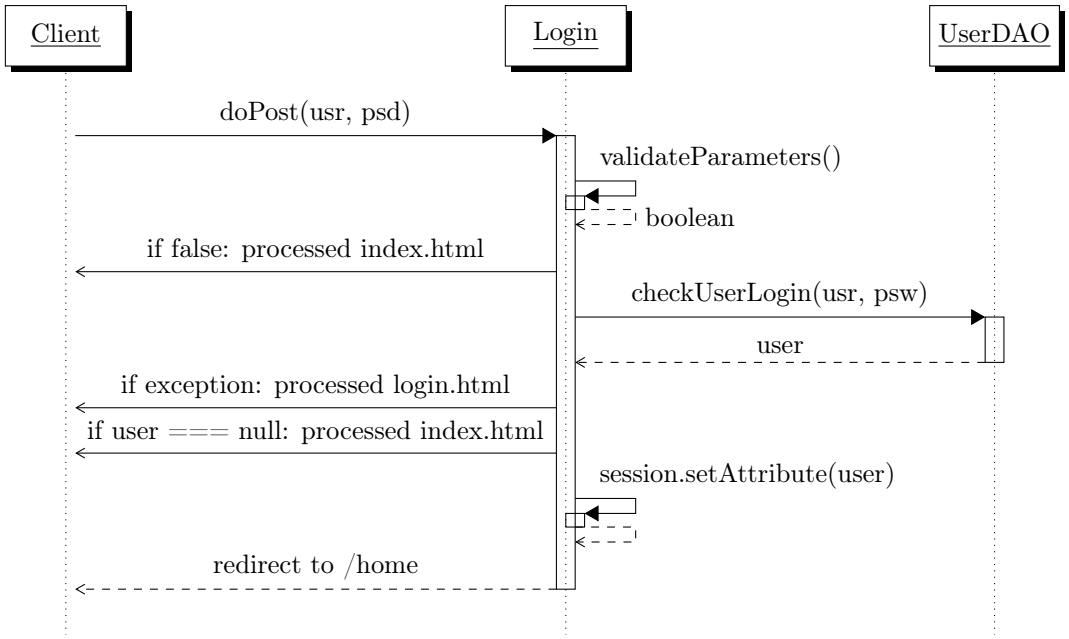
- Model Objects (Beans)
  - Account
  - Contact
  - Transaction
  - User
- Data Access Objects (Classes)

  - AccountDAO
  - ContactDAO
  - TransactionDAO
  - UserDAO
- Controllers (servlets)
  - AccountPage
  - AddContact
  - ErrorPage
  - HomePage
  - Login
  - Logout
  - MakeTransaction
  - ProfilePage
  - Registration
  - TransactionOutcomePage
- Filters
  - AlreadyLoggedInChecker
  - LoginChecker
- Utils
  - ConnectionHandler
  - ParameterValidator
  - PasswordHashing
- Views (Templates)
  - account.html
  - error.html
  - home.html
  - login.html
  - profile.html
  - registration.html
  - transaction-outcome.html
- Fragments (Templates)
  - navbar.html
  - sidebar.html

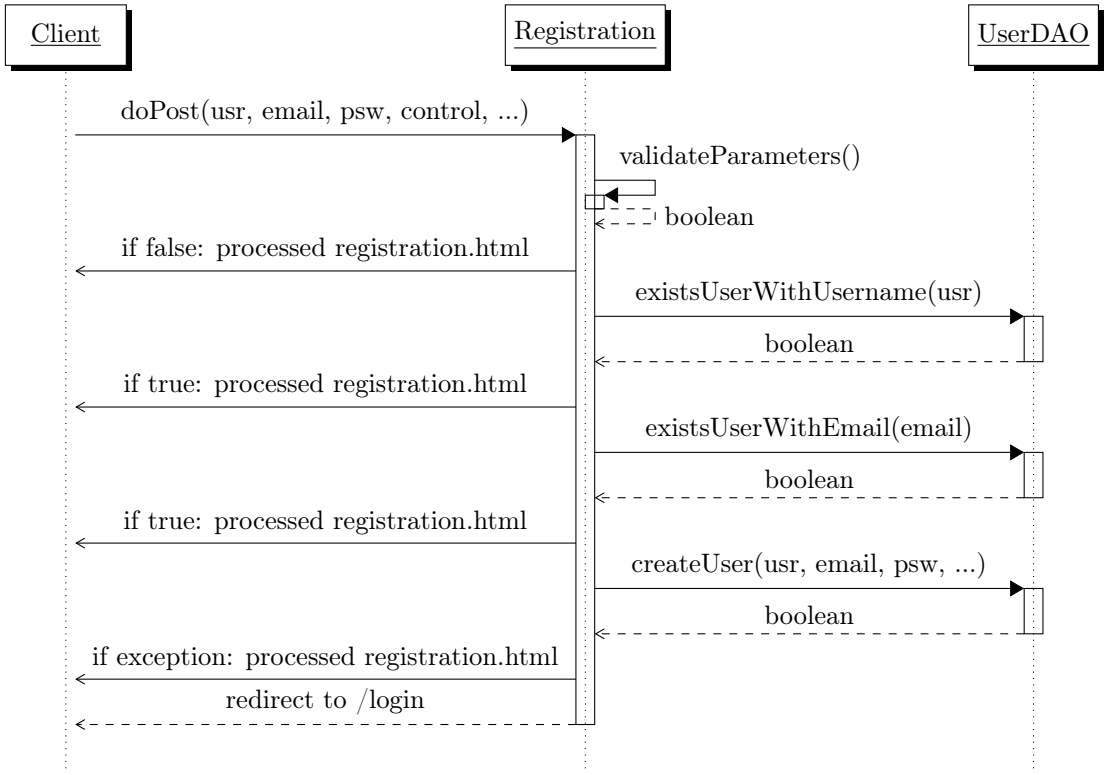
3.5 Eventi

Precisiazioni riguardo ai sequence diagram: con "processed" si intende il file HTML processato tramite `templateEngine.process()` dopo avere impostato le oppurtune variabili nel `context`; si assume che le get e le post di seguito rappresentate abbiano già superato i controlli presenti nei filtri.

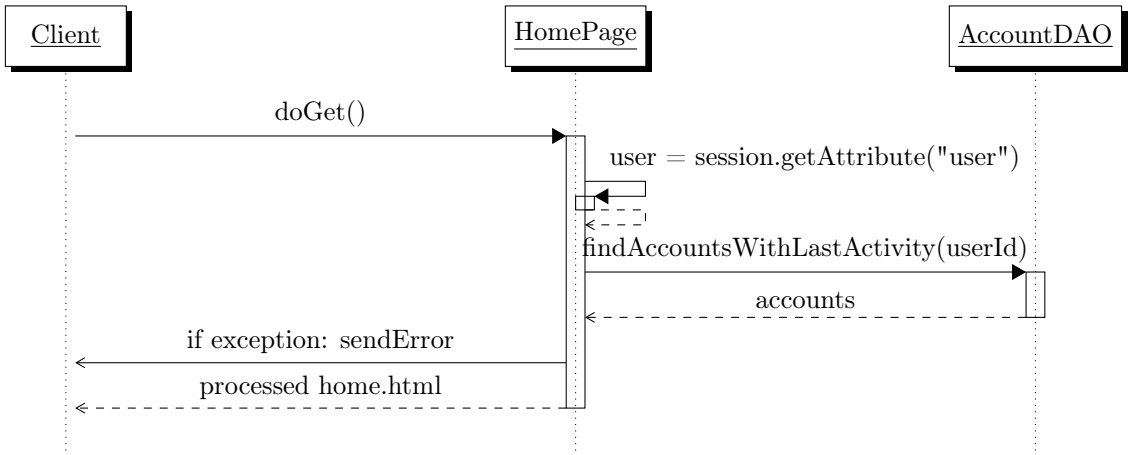
3.5.1 Login



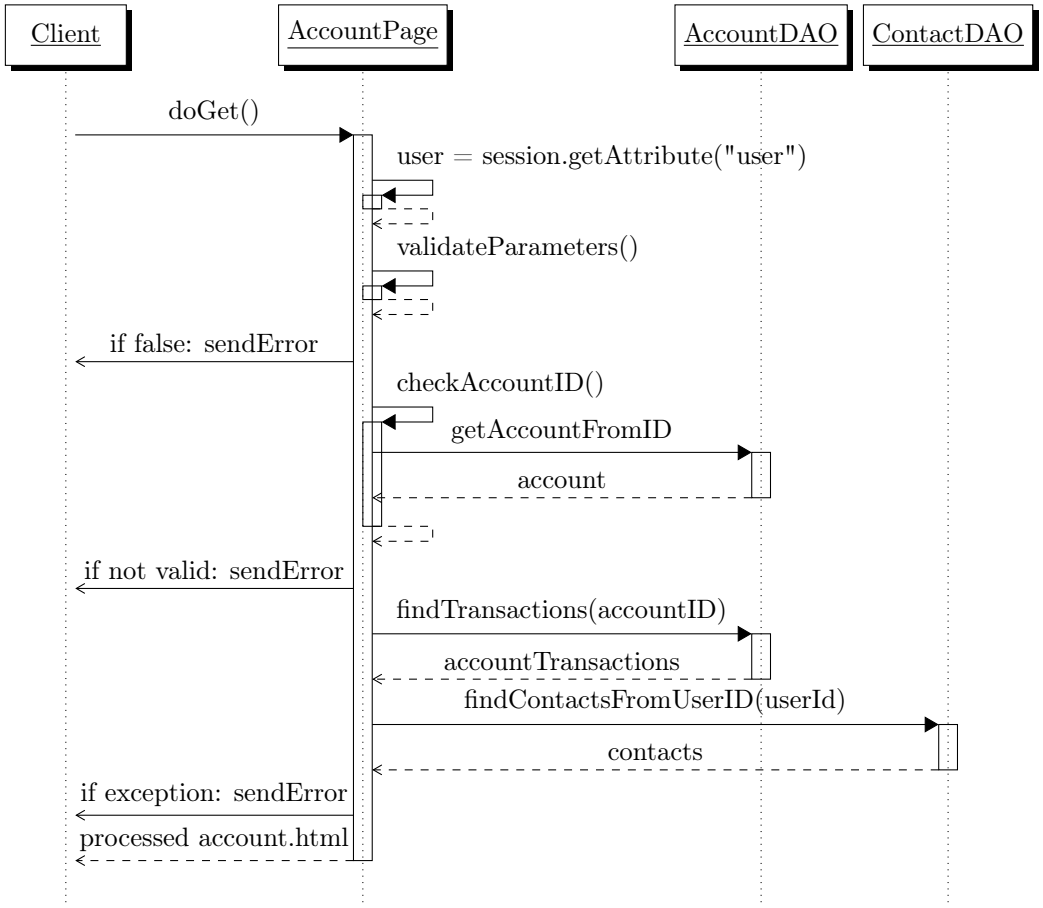
3.5.2 Registrazione



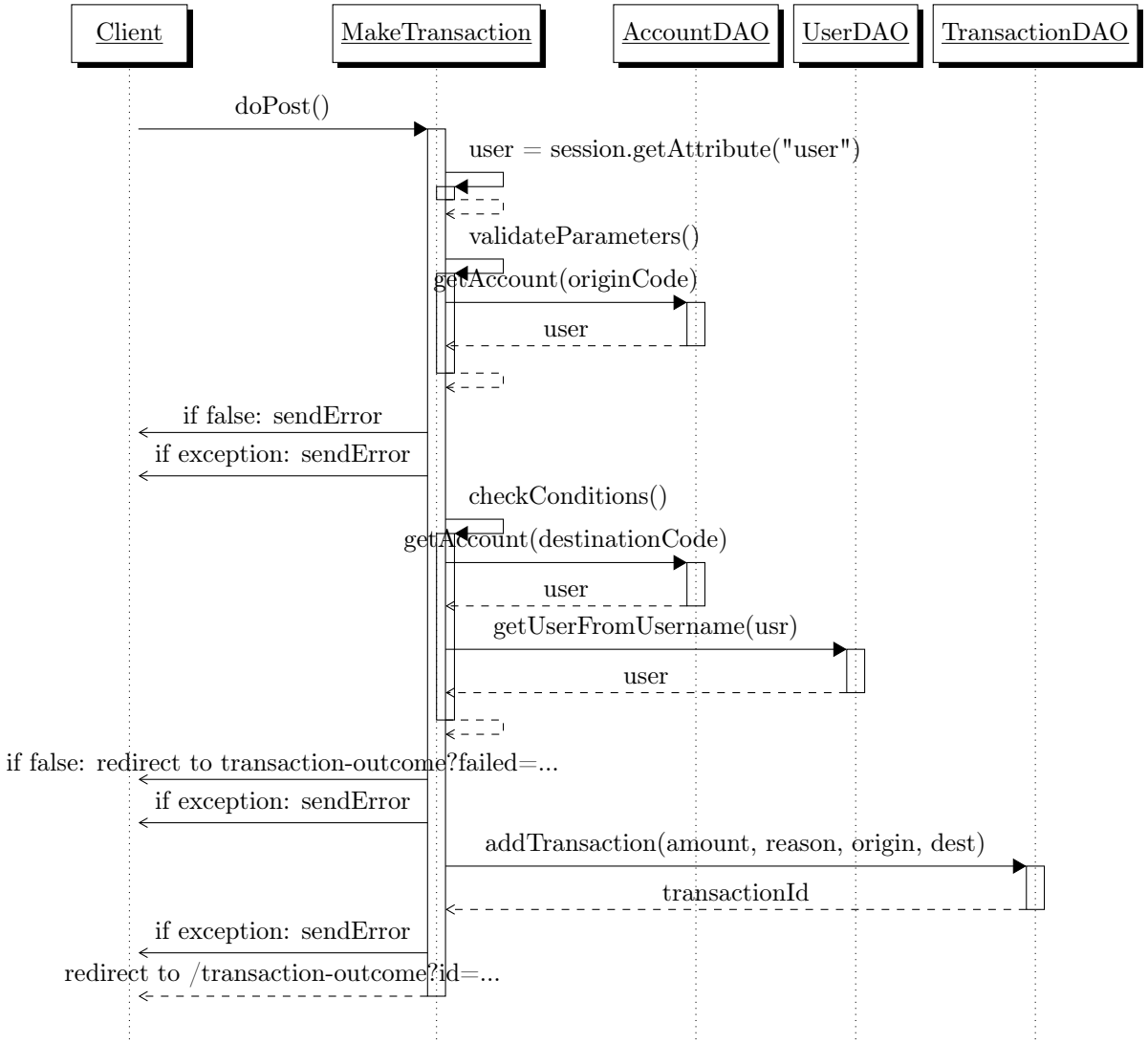
3.5.3 Accesso Home Page



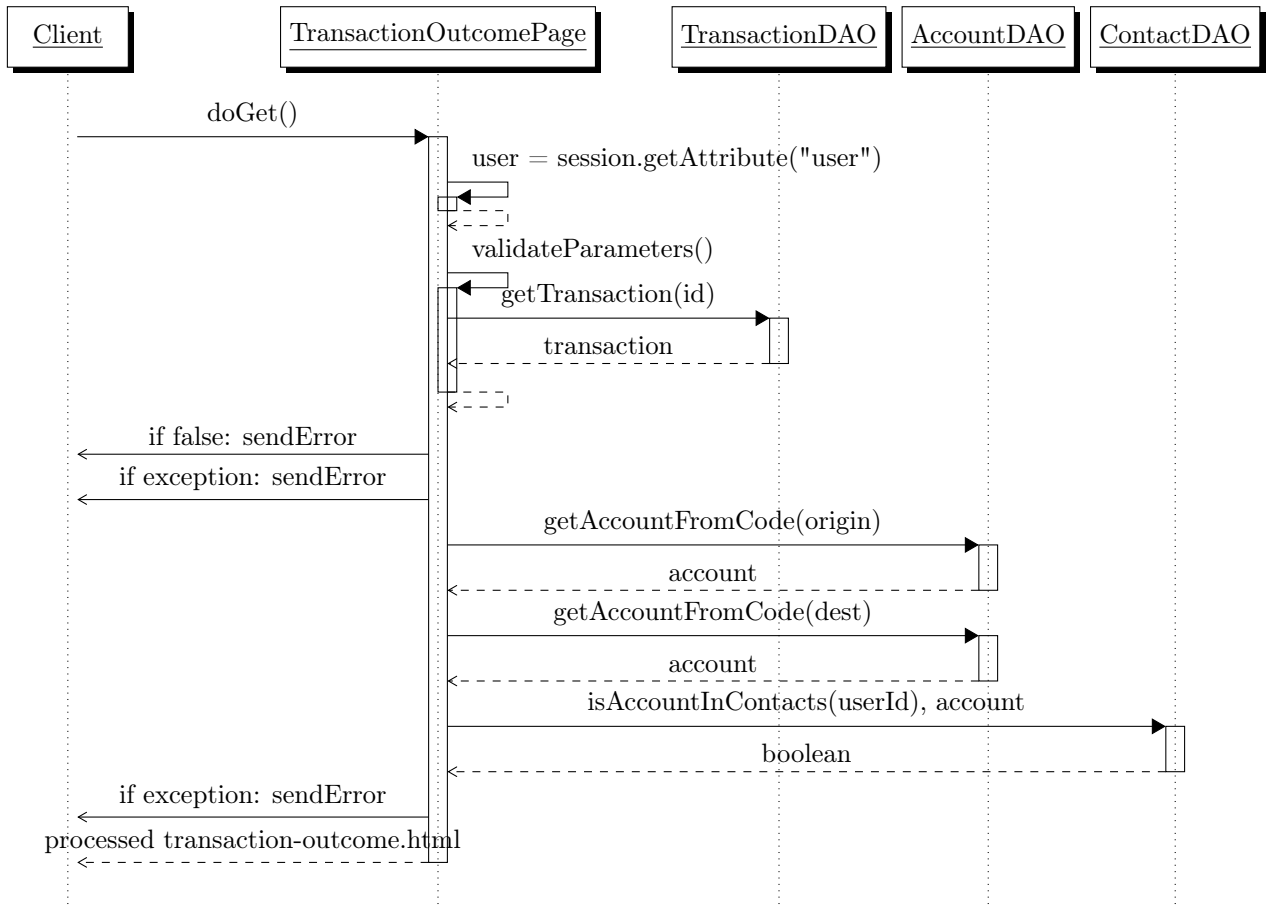
3.5.4 Accesso Account Page



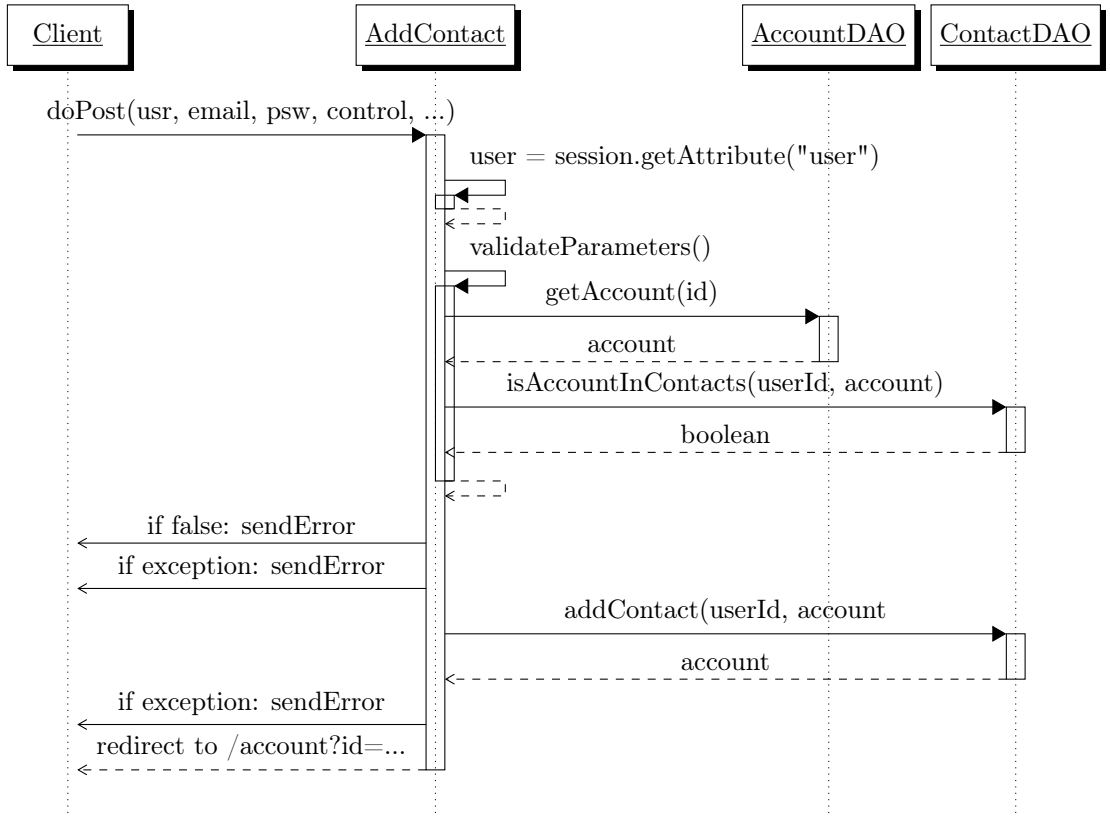
3.5.5 Esegui Transazione



3.5.6 Esito transazione



3.5.7 Aggiunta contatto



4 Versione RIA