

Progetto di Tecnologie Informatiche per il Web

Negri Riccardo

17 Giugno 2022



POLITECNICO
MILANO 1863

Docente: Fraternali Piero
Studente: Negri Riccardo 10729927 936820

Indice

1	Specifiche	3
1.1	Versione pure HTML	3
1.2	Versione RIA	4
2	Database design	5
2.1	Analisi testo delle specifiche	5
2.2	Progetto concettuale - Diagramma E-R (Entità-Relazione)	6
2.3	Progetto logico	6
2.4	Database schema	6
2.5	Hashing della password	8
2.6	Script Python per popolare il database	9
3	Versione pure HTML	10
3.1	Analisi requisiti delle specifiche	10
3.2	Completamento specifiche	10
3.3	Application design	11
3.4	Componenti	12
3.5	Eventi	13
3.5.1	Login	13
3.5.2	Registrazione	14
3.5.3	Accesso Home Page	14
3.5.4	Accesso Account Page	15
3.5.5	Esegui transazione	16
3.5.6	Esito transazione	17
3.5.7	Aggiunta contatto	18

4	Versione RIA	19
4.1	Analisi requisiti delle specifiche	19
4.2	Completamento specifiche	19
4.3	Application design	20
4.4	Componenti	21
4.5	Eventi e azioni	23
4.5.1	Login	24
4.5.2	Registrazione	25
4.5.3	Home View load	26
4.5.4	Account View load	27
4.5.5	Esegui transazione	28
4.5.6	Esito transazione	29
4.5.7	Aggiunta contatto	30
4.5.8	Auto completamento	31

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**.

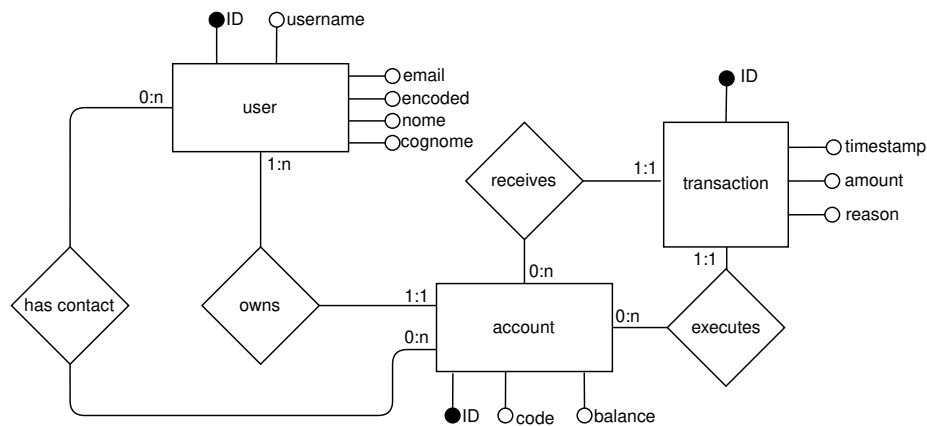
Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" 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

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 hashing delle password. Argon2 applica una funzione alla password di input insieme ad 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
```

Spiegazione 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 riportato impiega circa 400ms.

2.6 Script Python per popolare il database

Al fine di popolare automaticamente un nuovo database è stato scritto 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**.

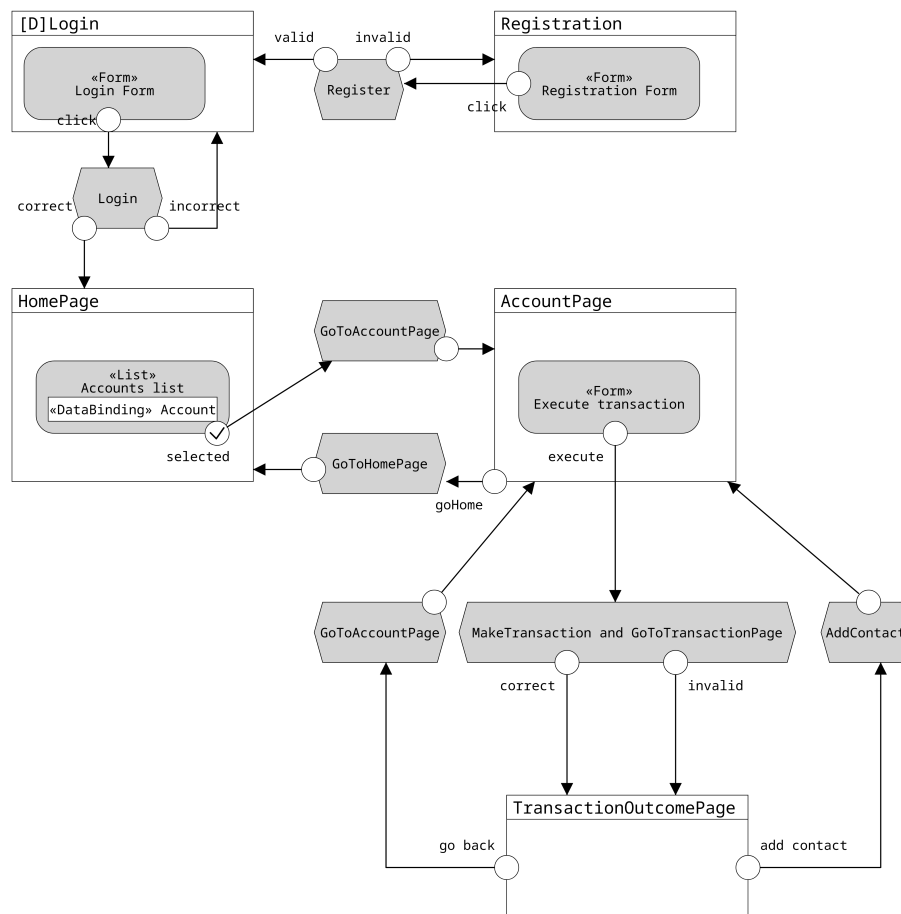
Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta **registrazione** e **login** mediante una pagina pubblica con opportune **form**. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i 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

- 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

- Si può accedere alla pagina precedente tramite la barra di navigazione laterale
- E' stata creata anche una pagina per visualizzare le informazioni dell'utente

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

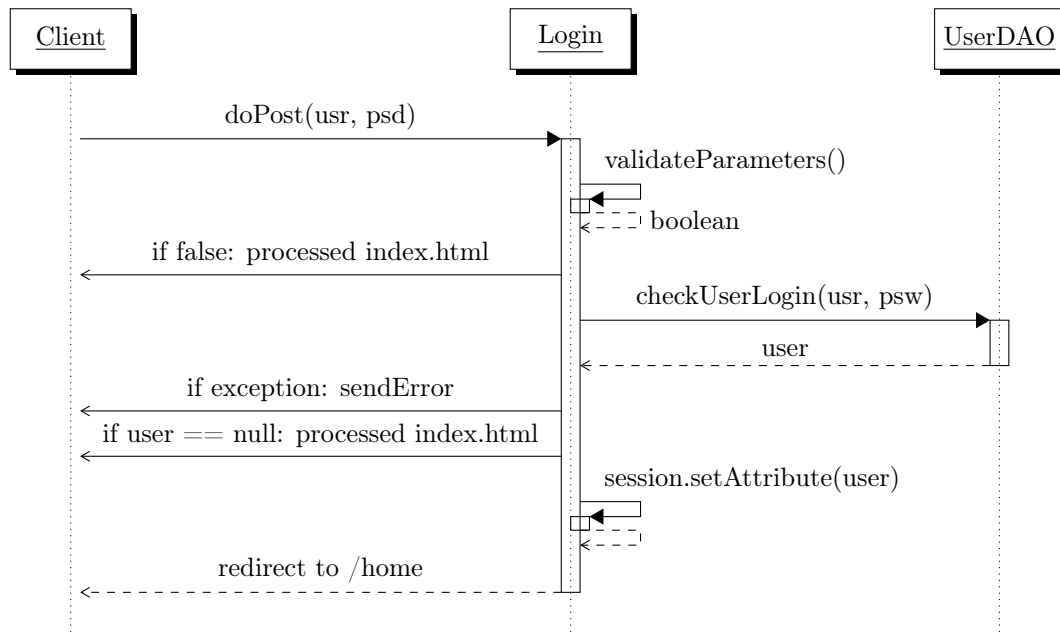
- Fragments (Templates)

- navbar.html
- sidebar.html

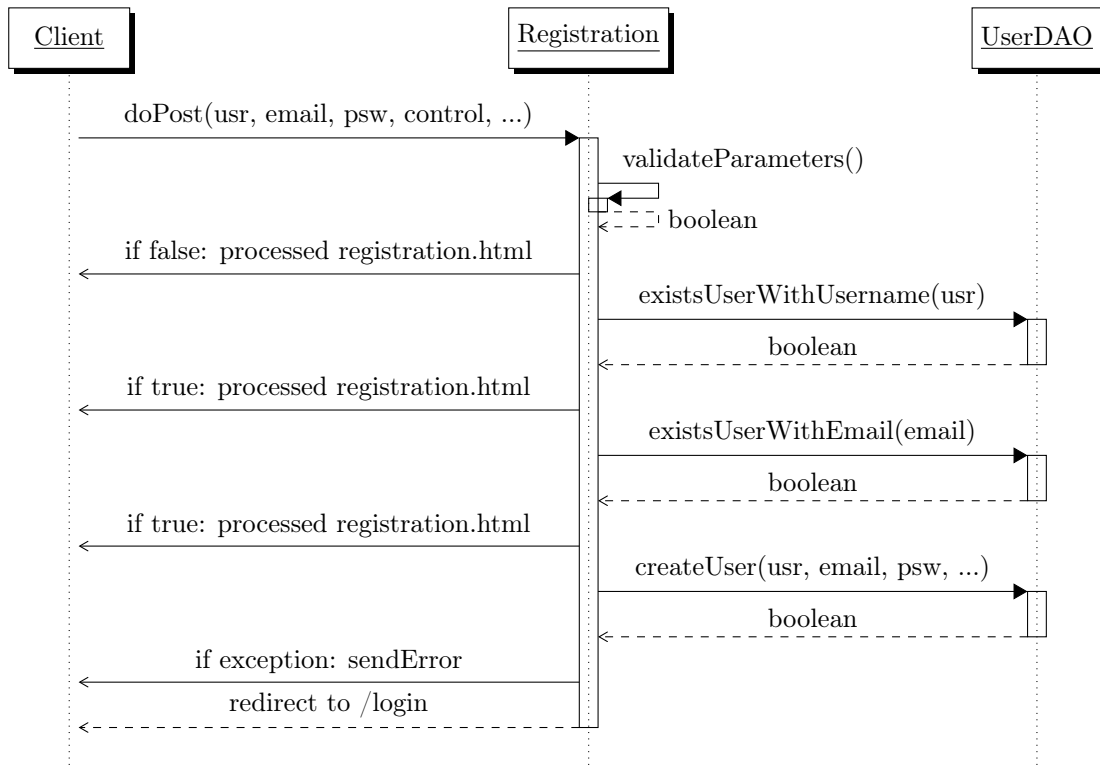
3.5 Eventi

Con "processed" si intende il file HTML processato tramite il metodo di Thymeleaf `templateEngine.process()` dopo avere impostato le opportune variabili nel `context`. Si assume che le get e le post di seguito rappresentate abbiano già superato i controlli presenti nei filtri.

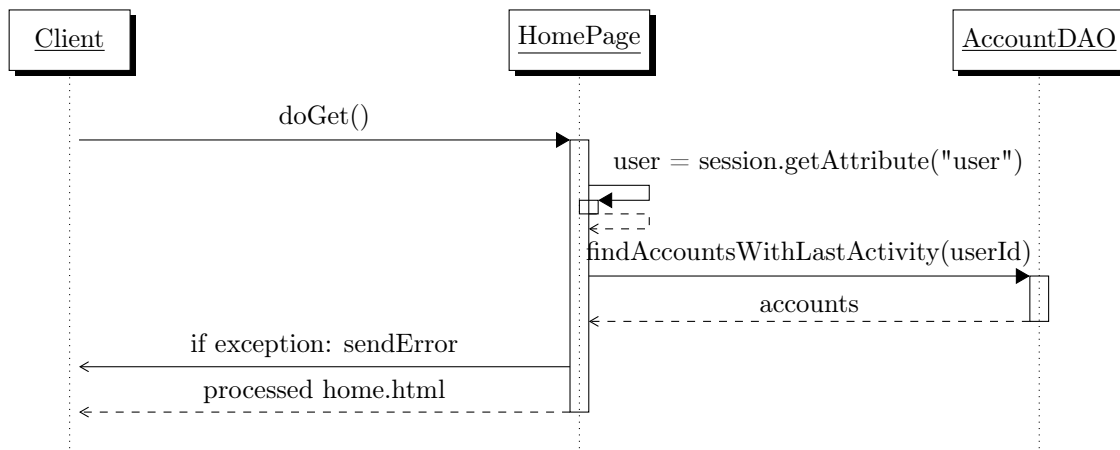
3.5.1 Login



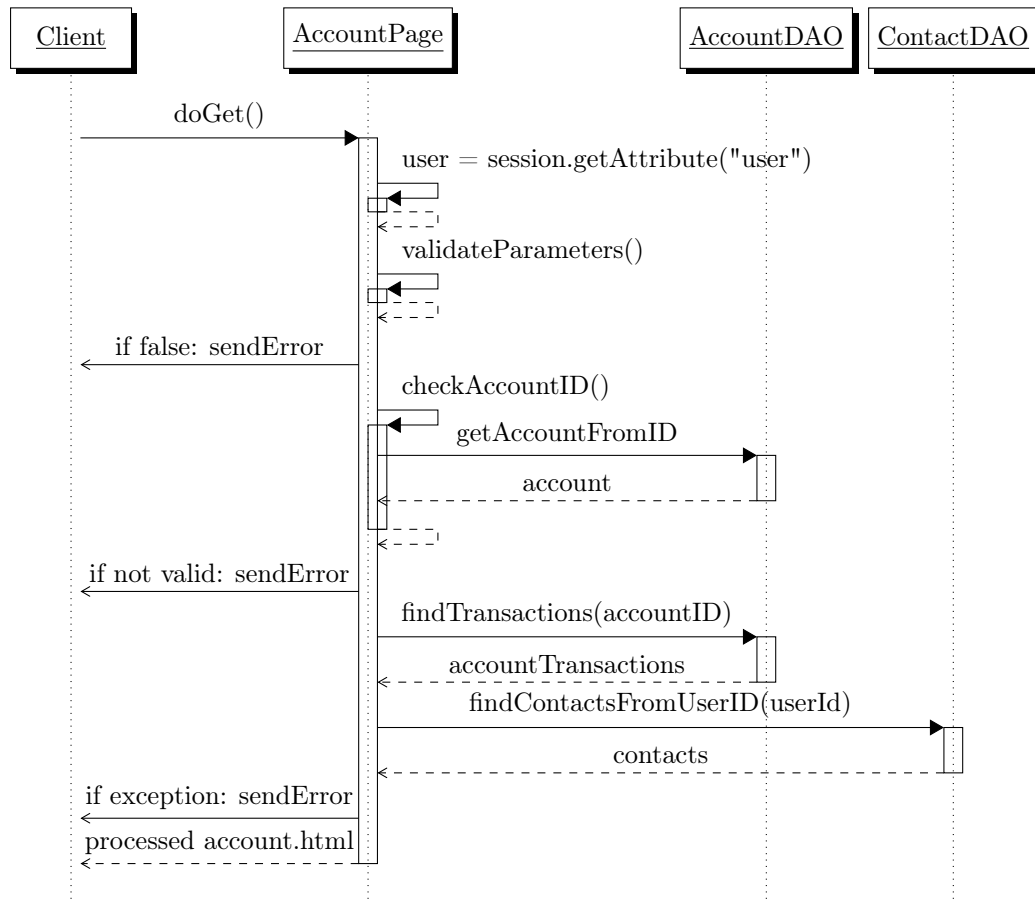
3.5.2 Registrazione



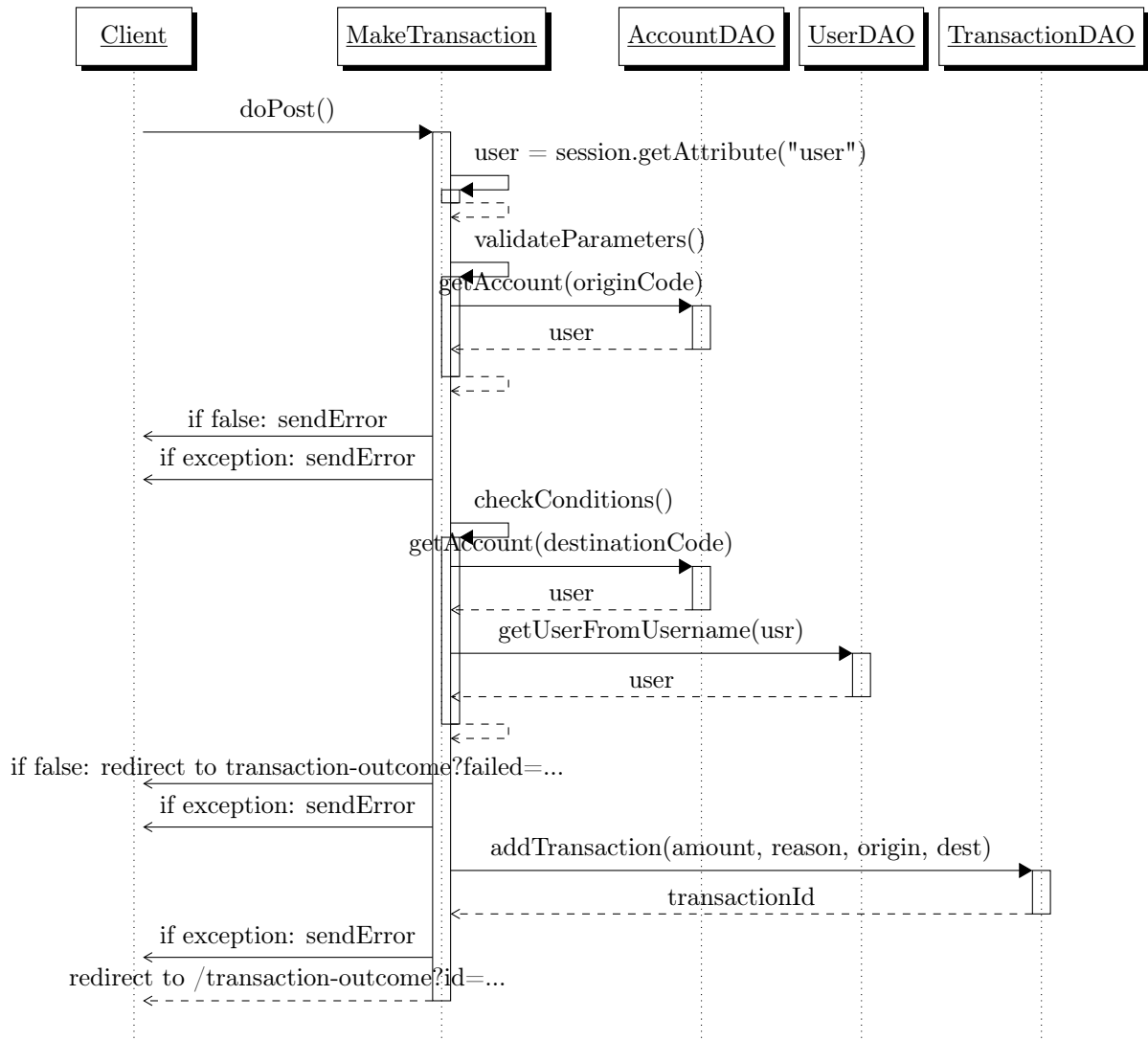
3.5.3 Accesso Home Page



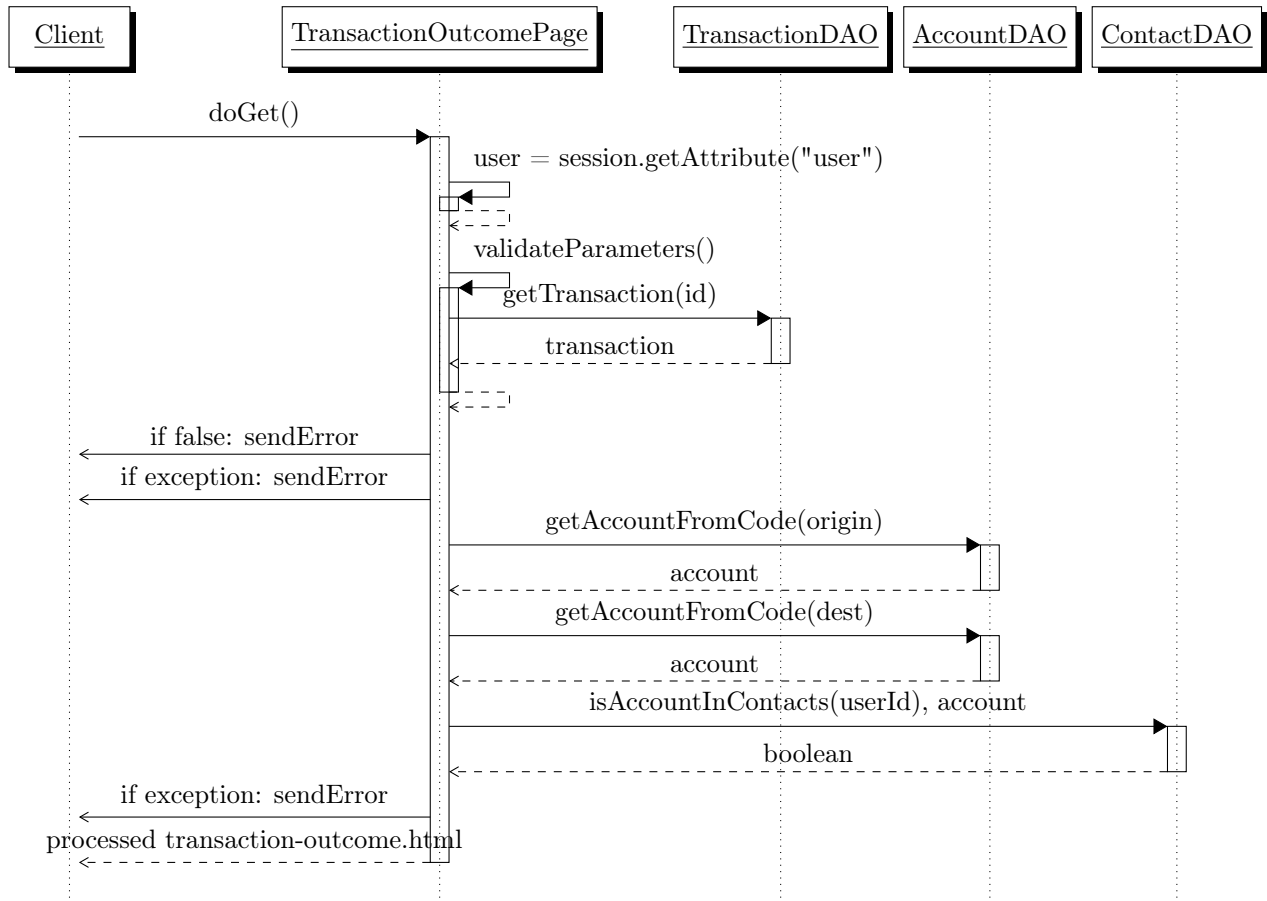
3.5.4 Accesso Account Page



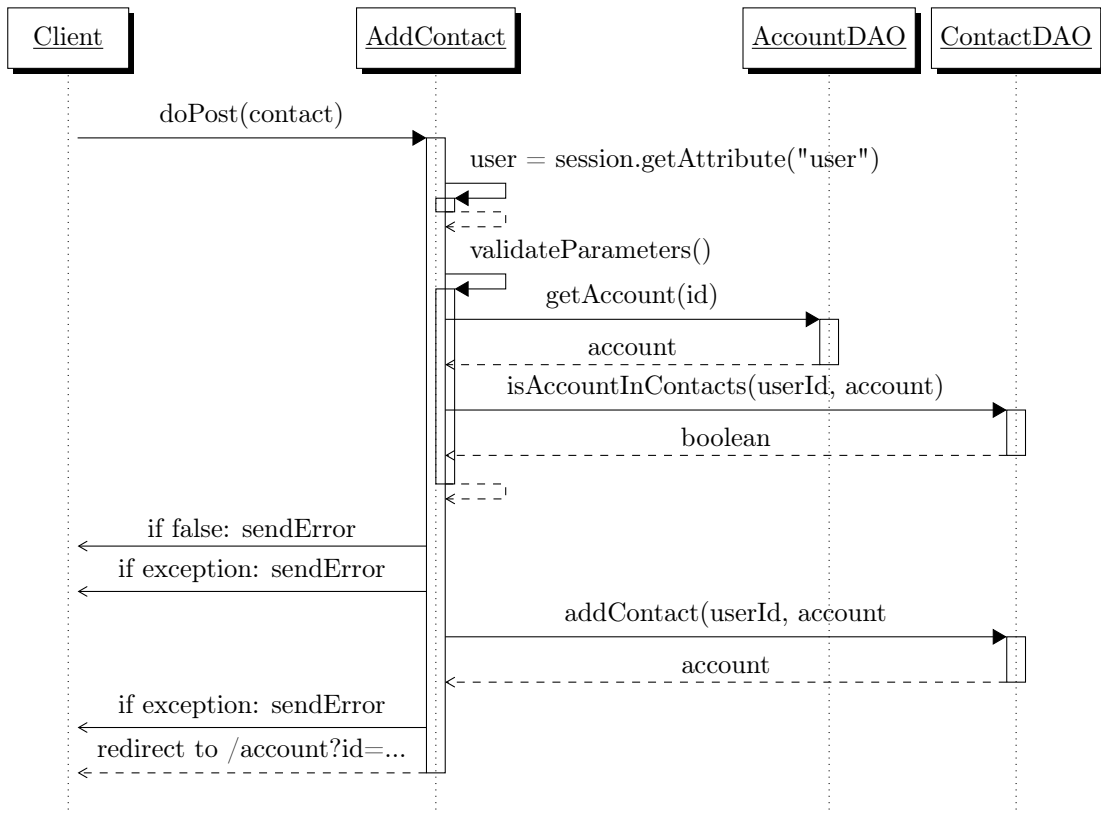
3.5.5 Esegui transazione



3.5.6 Esito transazione



3.5.7 Aggiunta contatto



4 Versione RIA

4.1 Analisi requisiti delle specifiche

Legenda: **pagine**, **view components**, **eventi**, **azioni**.

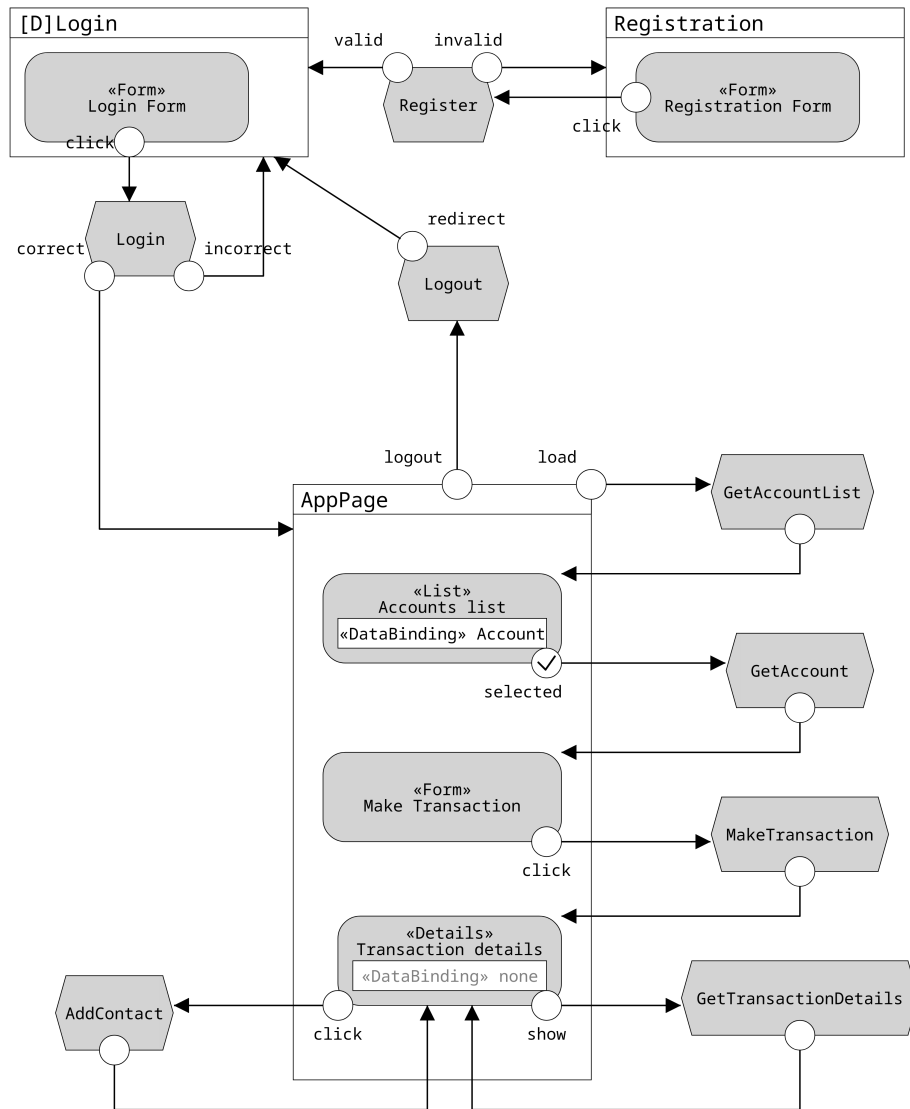
Si realizzi un'applicazione client server web che modifica le specifiche precedenti (subsection 3.1) come segue:

- 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.

4.2 Completamento specifiche

- Valgono i completamenti alla specifiche presenti nella versione pure HTML (subsection 3.2)
- L'auto-completamento propone inizialmente tutti i nomi utente nella propria rubrica e, una volta selezionato un nome valido, propone tutti i codici associati allo username selezionato.

4.3 Application design



4.4 Componenti

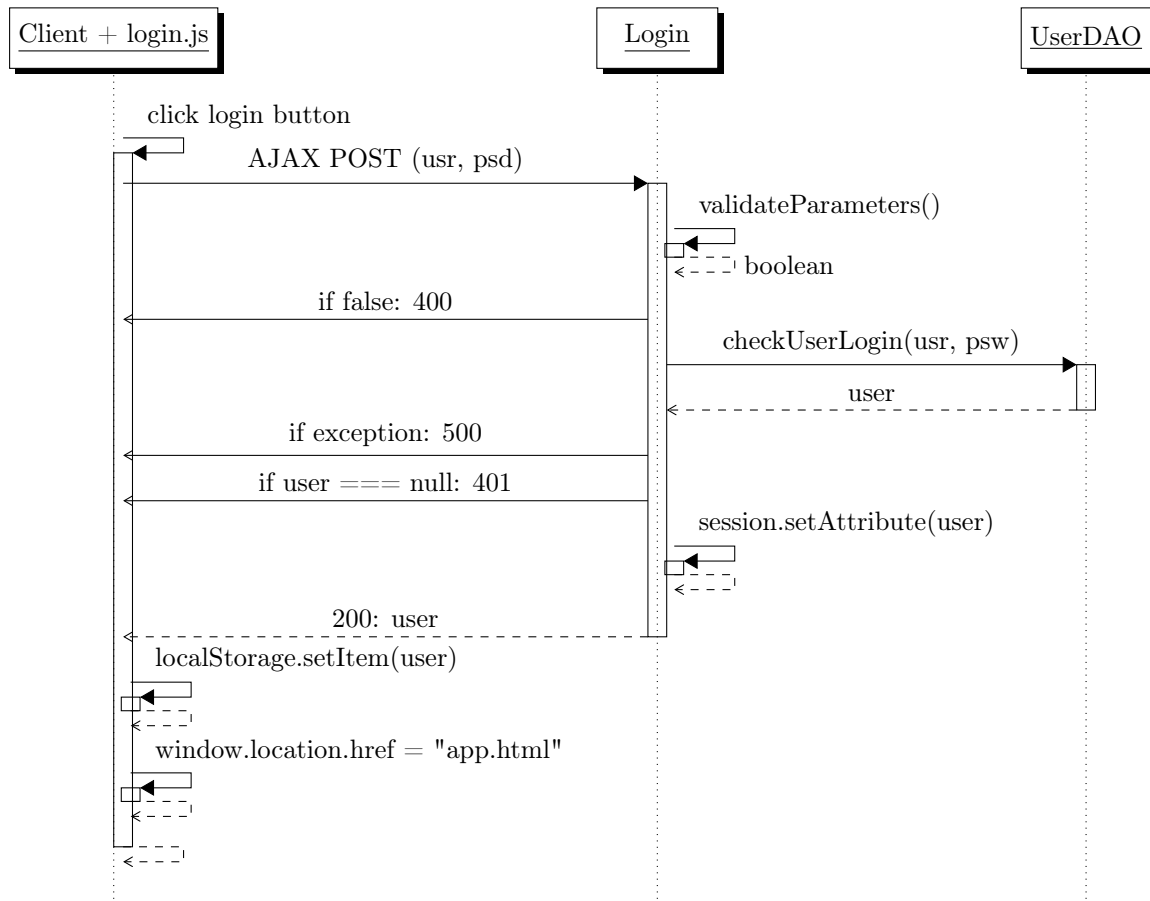
- Model Objects (Beans)
 - Account
 - Contact
 - Transaction
 - User
- Data Access Objects (Classes)
 - AccountDAO
 - ContactDAO
 - TransactionDAO
 - UserDAO
- Controllers (servlets)
 - AddContact
 - GetAccount
 - GetAccountList
 - GetTransactionDetails
 - Login
 - Logout
 - MakeTransaction
 - Registration
- Filters
 - AlreadyLoggedInChecker
 - LoginChecker
- Utils
 - ConnectionHandler
 - ParameterValidator
 - PasswordHashing
- Viste e componenti
 - app.html
 - * Navbar
 - * Sidebar
 - * Lista account

- * Lista transazioni
 - * Sezione saldo e informazioni account
 - * Form transazione
 - * Lista contatti
 - * Modal
- index.html
 - * Login form
- registration.html
 - * Registration form

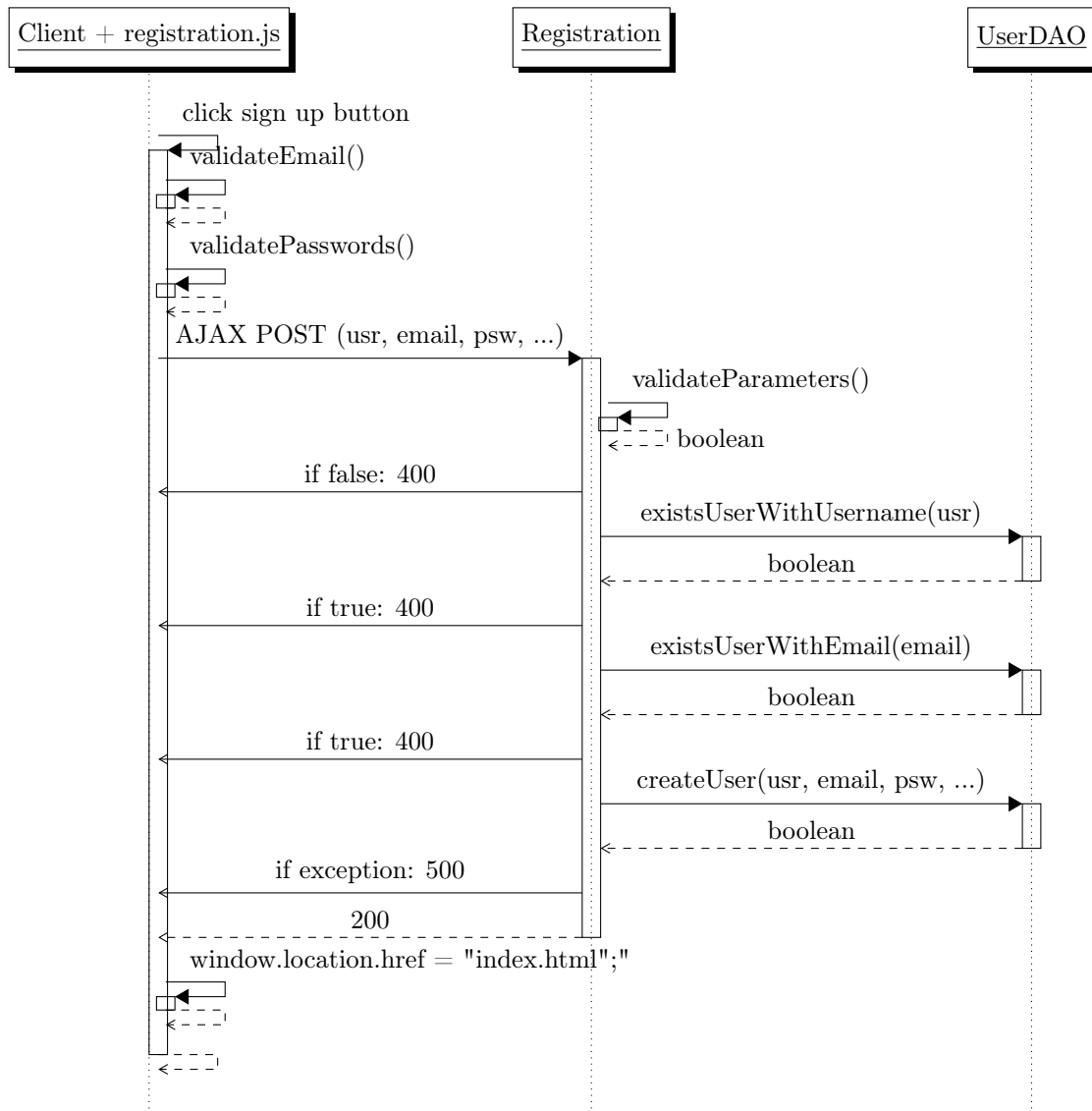
4.5 Eventi e azioni

Client side			Server side		
Evento	Azione	Controllore	Evento	Azione	Controllore
index → login form → submit	Controllo dati	function makeCall	iPOST usr psw	Controllo credenziali	Login
registration → registration form → submit	Controllo dati	function makeCall	POST usr email psw control name surname	Validazione e controllo dati e creazione utente	Registration
AppPage → load	Update view con lista account e update sidebar e navbar	function homeView	GET	Estrazione lista account dell'utente	GetAccount List
AppPage → select account	Update view con dettagli account e update sidebar e navbar	function accountView	GET accountId	Estrazione dettagli account dell'utente	GetAccount
AppPage → make transaction	Controllo validità parametri	function makeCall	POST origin destination amount reason username	Validazione e verifica parametri e condizioni necessarie. Esecuzione della transazione	Make Tran- saction
AppPage → show transaction outcome	Update view con modal con dettagli della transazione	function transaction- View	GET tran- sactionId	Estrazione dettagli transazione	Get Transac- tion Details
AppPage → add contact	Update view con messaggio di esito aggiunta contatto	function makeCall	POST account	Controllo parametri e aggiunta contatto a rubrica	Add Contact
AppPage → logout	Pulizia localStorage	function logout()	GET	Chiusura della sessione	Logout

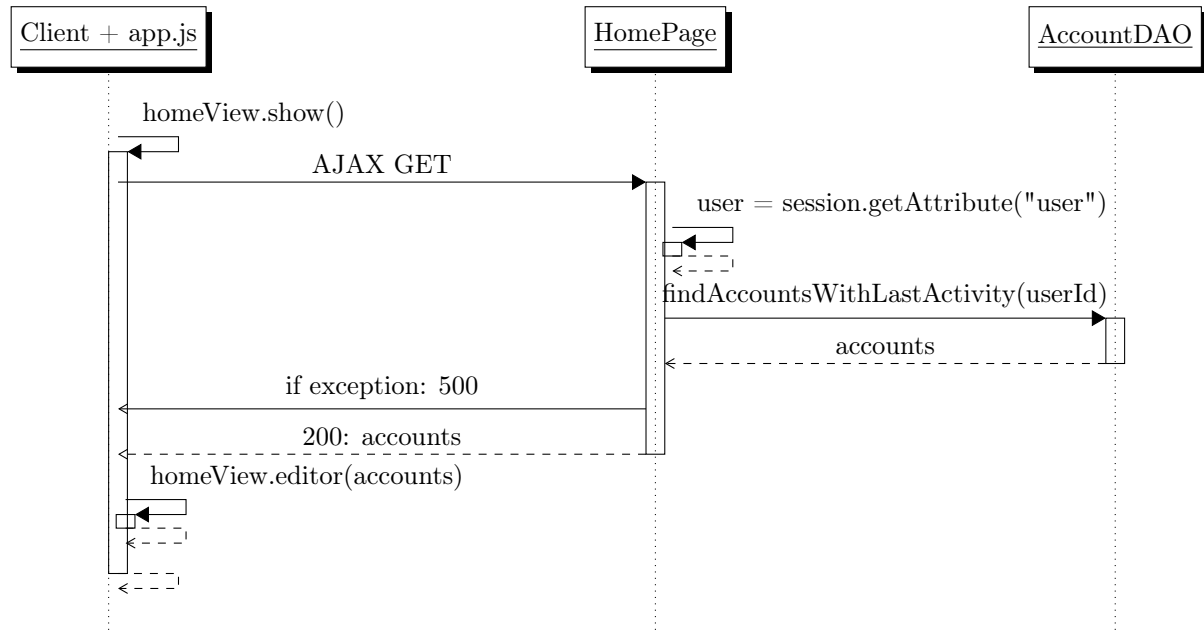
4.5.1 Login



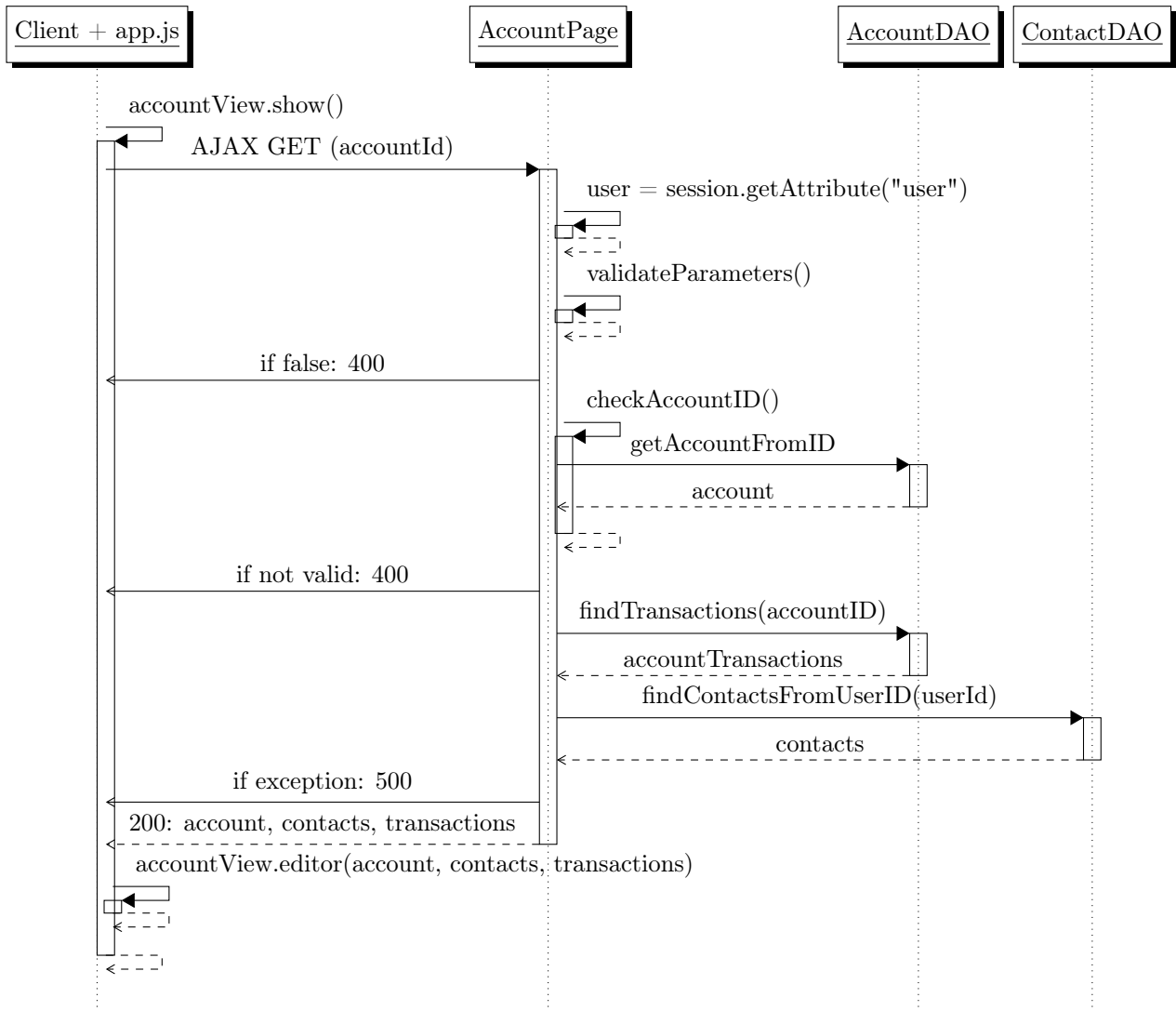
4.5.2 Registrazione



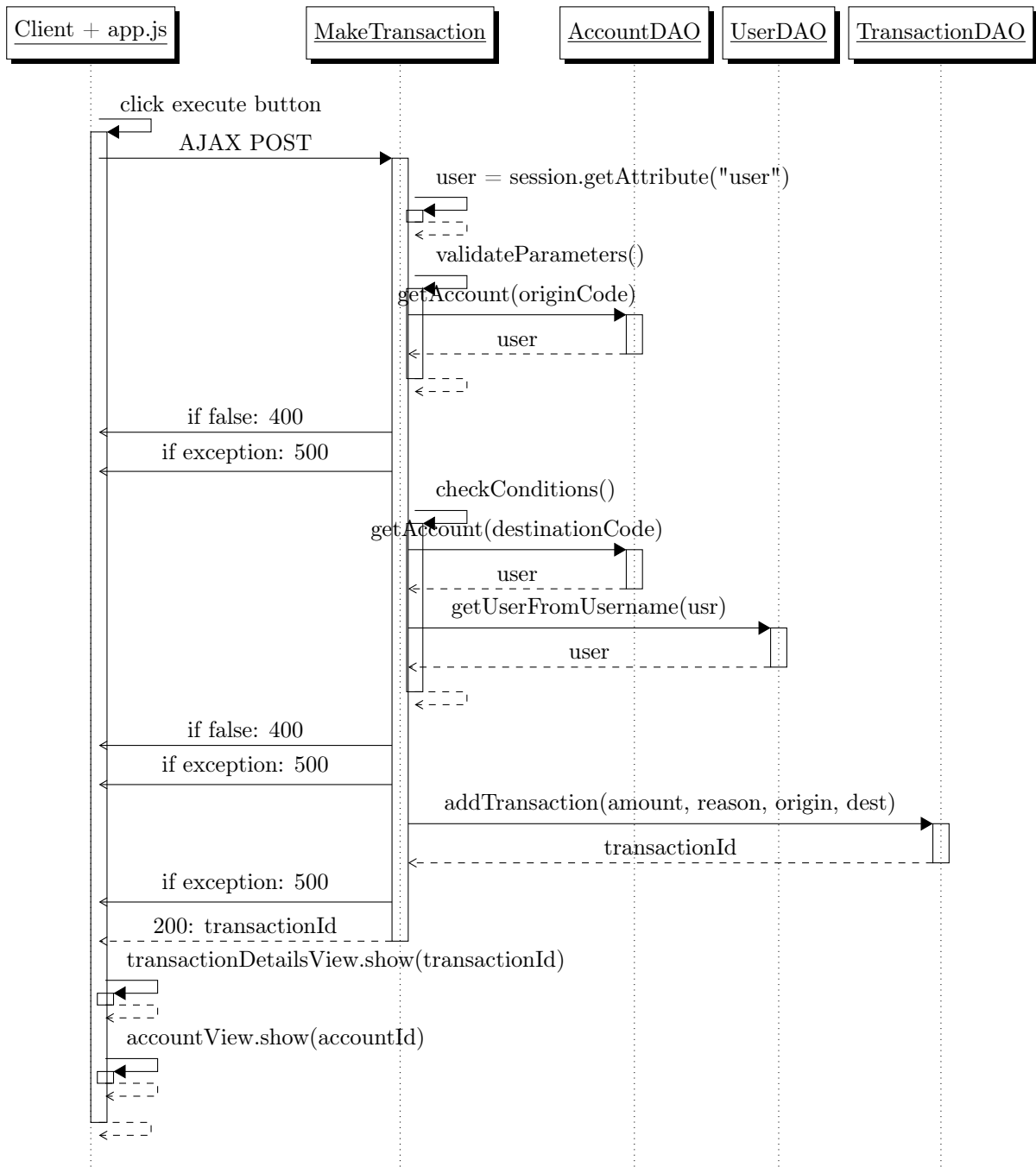
4.5.3 Home View load



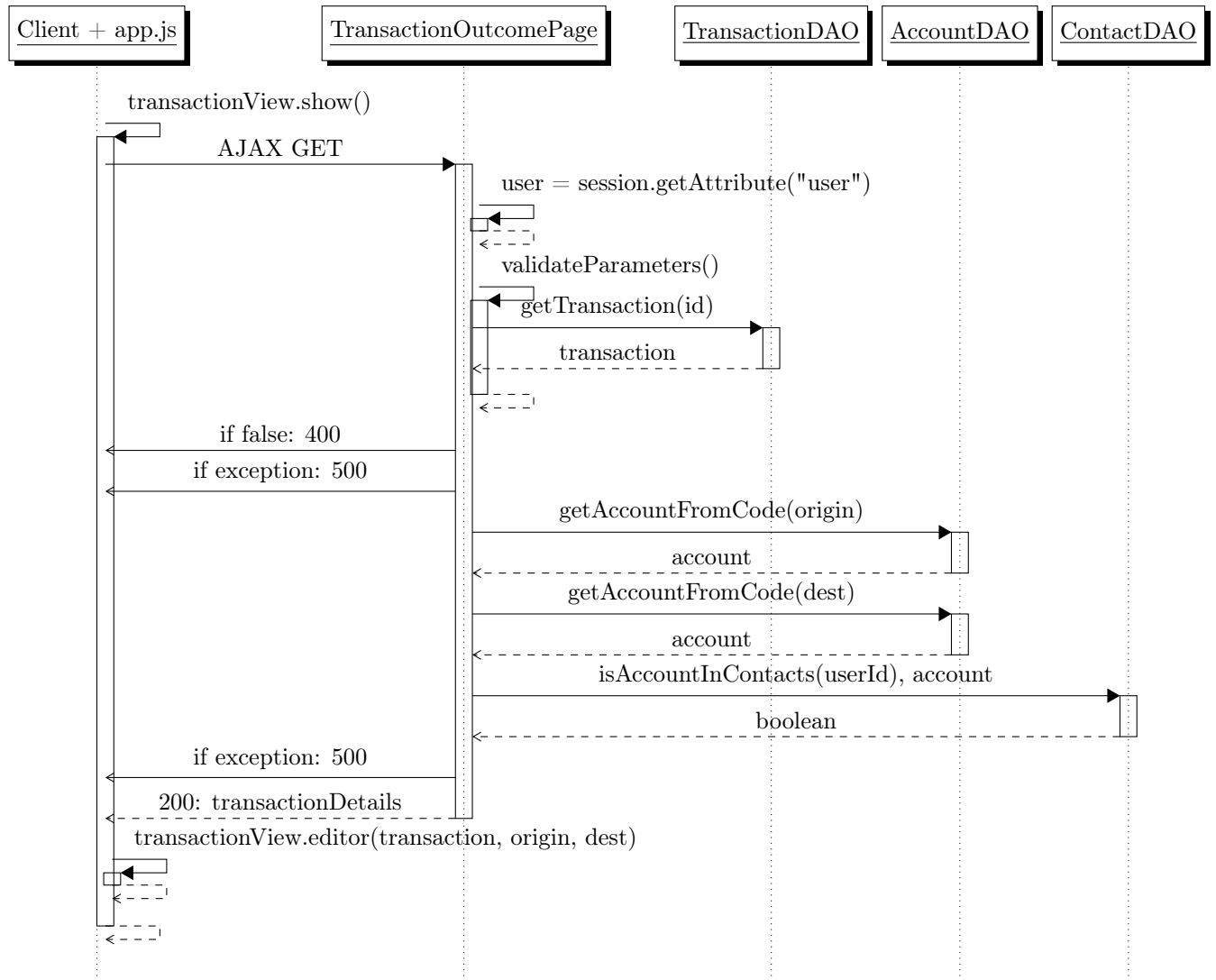
4.5.4 Account View load



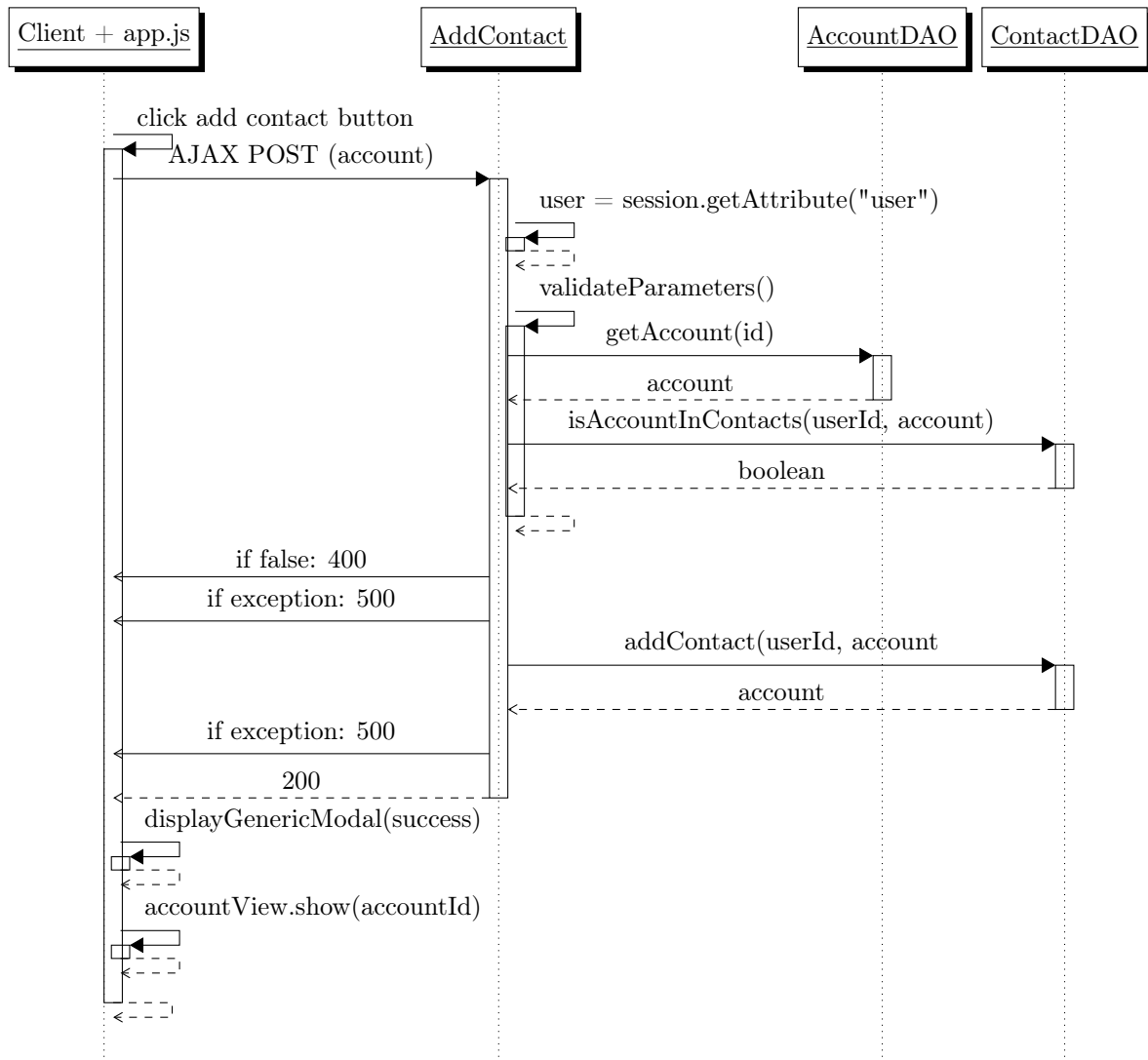
4.5.5 Esegui transazione



4.5.6 Esito transazione



4.5.7 Aggiunta contatto



4.5.8 Auto completamento

