de Donato Simone 10677578, Donno Valerio 10709082

Esercizio 4: Trasferimento di denaro

Versione RIA

Analisi dei dati

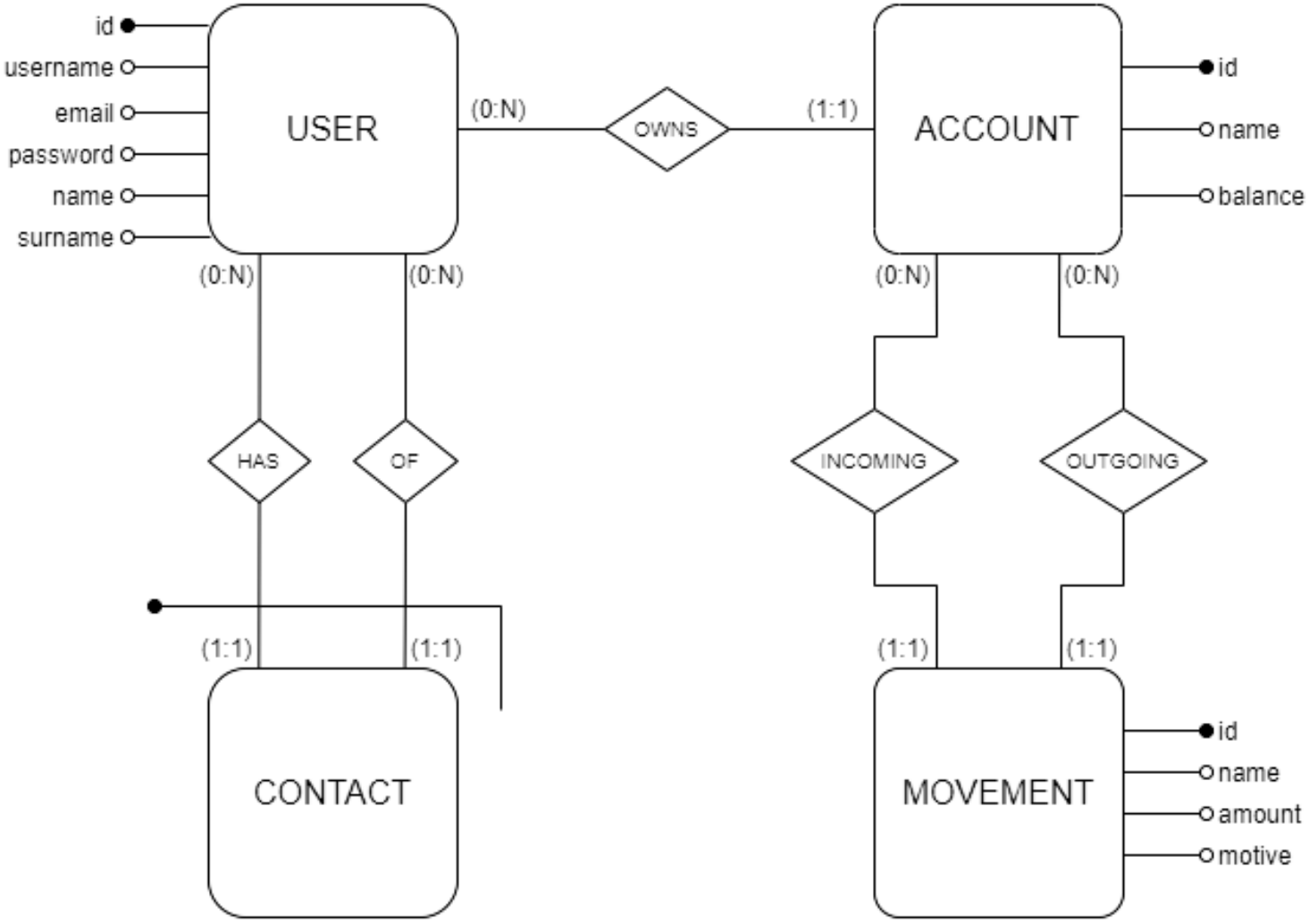
Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password"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

[Entities, attributes, relationships]

Schema ER database



L’entità “CONTACT” realizza la rubrica delle specifiche. La rubrica è pensata come una relazione N:M tra uno user e i suoi user in rubrica. Questa relazione N:M è tradotta tramite l’entità contact.

Schema logico database

```
CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email_UNIQUE` (`email`),
  UNIQUE KEY `username_UNIQUE` (`username`)
)
```

```
CREATE TABLE `account` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `balance` decimal(12,2) NOT NULL,
  `userid` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `userid_idx` (`userid`),
  CONSTRAINT `userid` FOREIGN KEY (`userid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `movement` (
  `id` int NOT NULL AUTO_INCREMENT,
  `date` timestamp NOT NULL,
  `amount` decimal(12,2) NOT NULL,
  `motive` varchar(200) NOT NULL,
  `inAccountID` int NOT NULL,
  `outAccountID` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `inAccountid_idx` (`inAccountID`),
  KEY `outAccountid_idx` (`outAccountID`),
  CONSTRAINT `inAccountid` FOREIGN KEY (`inAccountID`) REFERENCES `account` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `outAccountid` FOREIGN KEY (`outAccountID`) REFERENCES `account` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
CREATE TABLE `contact` (
  `ownerUserid` int NOT NULL,
  `contactUserid` int NOT NULL,
  PRIMARY KEY (`ownerUserid`,`contactUserid`),
  KEY `contactUserid_idx` (`contactUserid`),
  CONSTRAINT `contactUserid` FOREIGN KEY (`contactUserid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `ownerUserid` FOREIGN KEY (`ownerUserid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
```

Analisi requisiti dell'applicazione

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 *(i.e. invio della form)*. 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.

[Pages (views), view components, events, actions]

Completamento delle specifiche

- La pagina di login contiene form sia per la registrazione che per il login. Le due vengono mostrate all'utente tramite un pulsante login/signup che cambiano la pagina senza interpellare il server
- L'invio del form di registrazione causa, oltra alla creazione di un nuovo utente, anche l'autenticazione, e l'utente è portato alla pagina HOME
- L'azione di logout riporta l'utente alla pagina di LOGIN
- La pagina HOME contiene un form per creare un nuovo conto all'invio della form con un saldo iniziale arbitrario positivo
- Qualora si tentasse l'accesso alla home non avendo prima effettuato il login, si verrà reindirizzati alla pagina di LOGIN.

Sommario

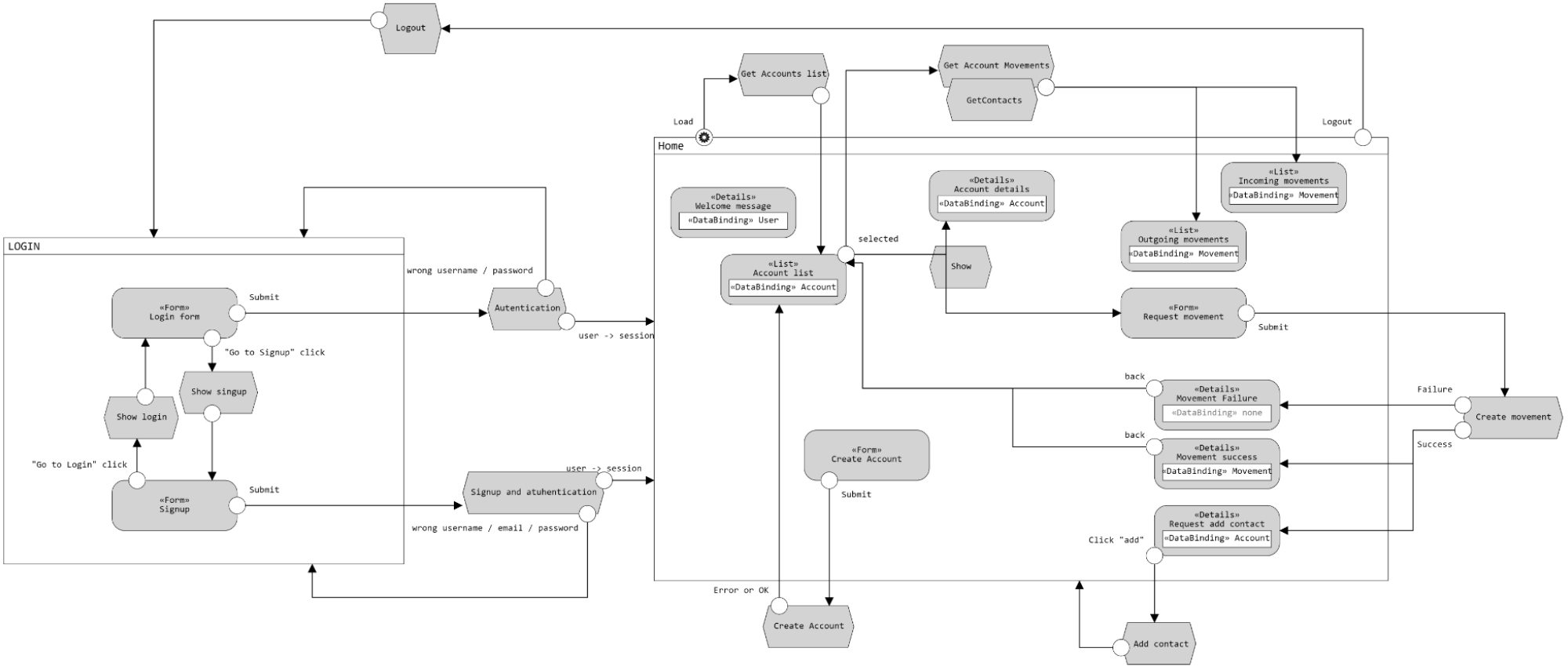
Pagine e componenti

- LOGIN
  - Login Form
  - Signup Form
- HOME
  - Messaggio di welcome
  - Lista account
  - Form creazione account
  - Dettaglio account (trasferimenti in uscita e ingresso)
  - Form creazione trasferimento
  - Messaggio fallimento trasferimento
  - Messaggio conferma trasferimento
  - Messaggio di richiesta aggiunta in rubrica

Eventi e azioni

- Login
  - Autenticazione
- Signup
  - Creazione utente
  - Autenticazione
- Click su tasto login/signup
  - Mostra form di login/signup
- Invio form creazione account
  - Creazione account
- Click su conto
  - Mostrare dettagli (trasferimenti e form trasferimento)
- Invio form trasferimento
  - Creazione trasferimento
- Click su Aggiungi contatto dopo trasferimento
  - Aggiungi contatto a rubrica
- Scrivi numero in campo id utente nel form trasferimento
  - Mostra suggerimento per autocompletamento
- Scrivi numero in campo id account
  - Mostra suggerimento tra account di utente inserito in campo id utente (se in rubrica)

IFML Design


Logout
Get Accounts list
Get Account Movements
GetContacts
Load
Logout
Home
«Details»
Welcome message
«DataBinding» User
«Details»
Account details
«DataBinding» Account
«List»
Incoming movements
«DataBinding» Movement
«List»
Outgoing movements
«DataBinding» Movement
selected
«List»
Account list
«DataBinding» Account
Show
«Form»
Request movement
Submit
back
«Details»
Movement Failure
«DataBinding» none
Failure
Create movement
Success
back
«Details»
Movement success
«DataBinding» Movement
«Form»
Create Account
Submit
«Details»
Request add contact
«DataBinding» Account
Click "add"
Error or OK
Create Account
Add contact
LOGIN
wrong username / password
«Form»
Login form
Submit
Autentication
user -> session
"Go to Signup" click
Show singup
Show login
"Go to Login" click
user -> session
«Form»
Signup
Submit
Signup and atuhentication
wrong username / email / password

Eventi e azioni

Client side		Server side	
Evento	Azione	Evento	Azione
Login-> login form -> submit	Controllo dati	POST (username, password)	Controllo credenziali e autenticazione
Login-> signup form -> submit	Controllo dati	POST (username, pass, email, nome, cognome)	Signup e autenticazione
Login-> Click “login”/”signup”	Mostra form di login/signup	—----------	—----------
Home -> load	Aggiorna e mostra lista account	GET (nessun parametro)	Estrazione account utente
Home -> create account form -> submit	Controllo dati	POST (nome account, balance)	Creazione account
Home -> click su “stato account”	Aggiorna e mostra lista moviment, aggiorna lista contattii	GET (accountid) e GET (nessun parametro)	Estrazione movimenti in e out di account, estrazione contatti di user
Home -> request movement form -> submit	Controllo dati	POST (userid, accountid, amount, motive, thisaccountid[hidden])	Creazione movimento
Home -> movement success e user non tra contatti	Mostra richiesta di aggiunta tra i contatti	—----------	—----------
Home -> Click su “aggiungi contatto”	Aggiunge contatto a lista per autocomplete	POST (contactUserid) (ownerUserid non serve, è in sessione)	Creazione contatto
Home -> Scrivere in campo “user id” in form richiesta movimento	Proposta di autocompletamento con user tra i contatti	—----------	—----------
Home -> Click su campo “account” in form richiesta movimento	Proposta di autocompletamento con account di user inserito	—----------	—----------

Controller / event handler

Client side		Server side	
Evento	Controllore	Evento	Controllore (servlet)
Login-> login form -> Submit	Function makeCall	POST (username, password)	CheckLogin
Login-> signup form -> Submit	Function makeCall	POST (username, pass, email, nome, cognome)	CheckSignup
Login-> Click “login”/”signup”	Function “EventListener”	—----------	—----------
Home -> Load	Function PageOrchestrator start e refresh	GET (nessun parametro)	GetAccountsData
Home -> Create account form -> submit	Function makeCall	POST (nome account, balance)	CreateAccount
Home -> Click su “stato account”	Function StatusView.show (fa makeCall)	GET (accountid) e GET (nessun parametro)	GetMovementData GetContactsData
Home -> Request movement form -> submit	Function makeCall	POST (userid, accountid, amount, motive, thisaccountid[hidden])	RequestMovement
Home -> Movement success e account non tra contatti	Function ResultView.show	—----------	—----------
Home -> Click su “aggiungi contatto”	Function MakeCall	POST (userid) (thisuserid non serve è in sessione)	AddContact
Home -> Scrivere in campo “user id” in form richiesta movimento	Function “EventListener” (in StatusView)	—----------	—----------
Home -> Click su campo “account” in form richiesta movimento	Function “EventListener” (in StatusView)	—----------	—----------

Server side components

Model object (Beans)

- User
- Account
- Movement
- UserContacts

Data Access Objects (Classes)

- UserDAO
  - checkCredentials(username, passwrod)
  - registerUser(name, surname, username, email, password)
  - getUserByID(userID)
  - getUserByUsername(username)
  - getUserByEmail(email)
  - addContact(ownerUserID, contactUserID)
  - hasContact(ownerUserID, contactUserID)
  - getContacts(ownerUserID)
- AccountDAO
  - getAccountsByUser(userID)
  - getAccountByID(accountID)
  - getAccountByName(name)
  - createAccount(userID, balance)
- MovementDAO
  - getIncomingMovementsByAccount(accountID)
  - getOutgoingMovementsByAccount(accountID)
  - getMovementByID(movementID)
  - requestMovement(amount, date, reason, inAccount, outAccount)

(N.B.: Il contact non ha un DAO in quanto è semplicemente una mappa di User e account posseduta da uno User.)

Controllers (Servlets)

- CheckLogin
- CheckSignup
- GetAccountsData
- CreateAccount
- GetMovementsData
- RequestMovement
- GetContactsData
- AddContact
- Logout

Client side components

Login

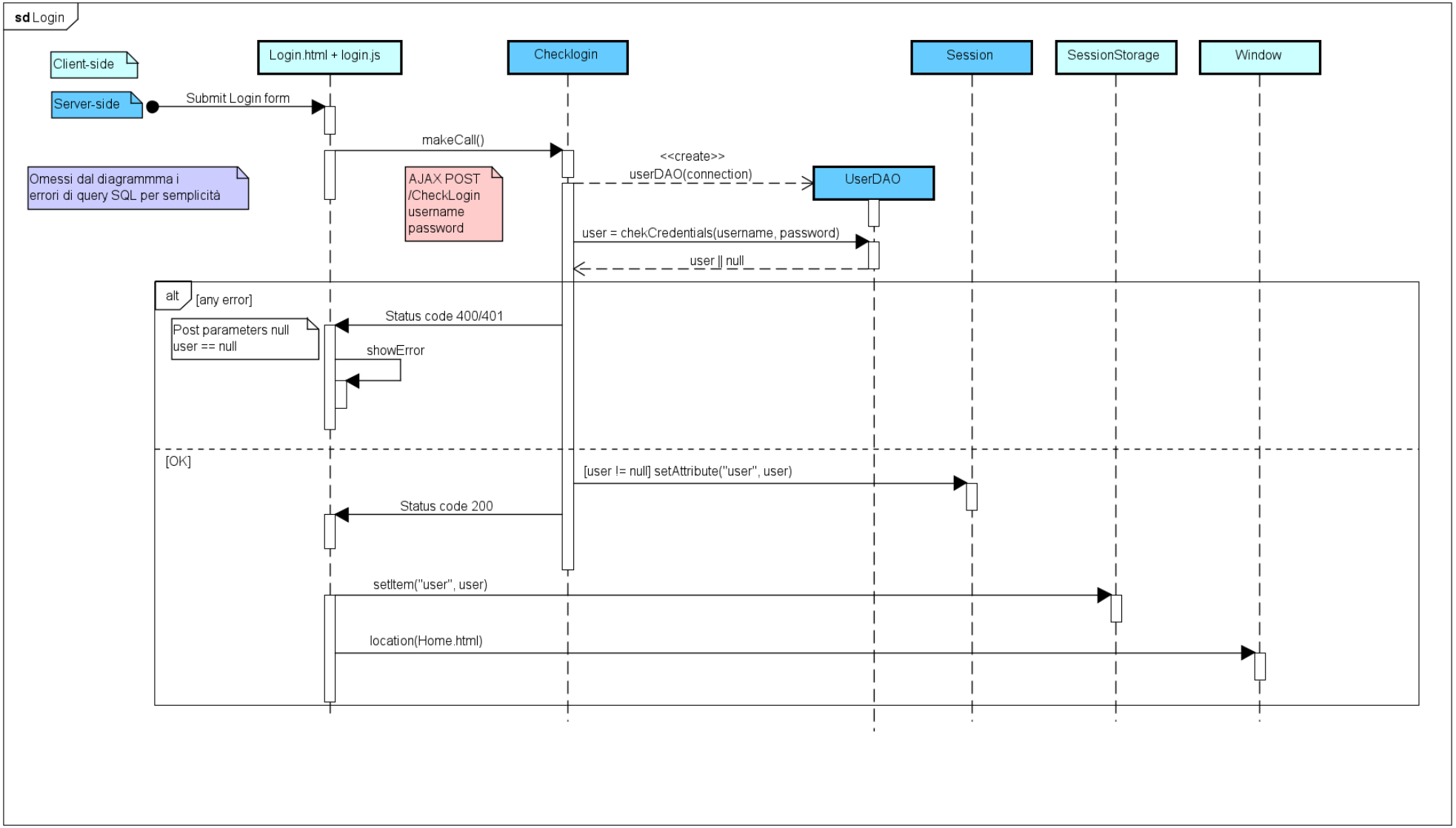
- Login form
  - Gestione submit ed errori
- Signup form
  - Gestione submit ed errori

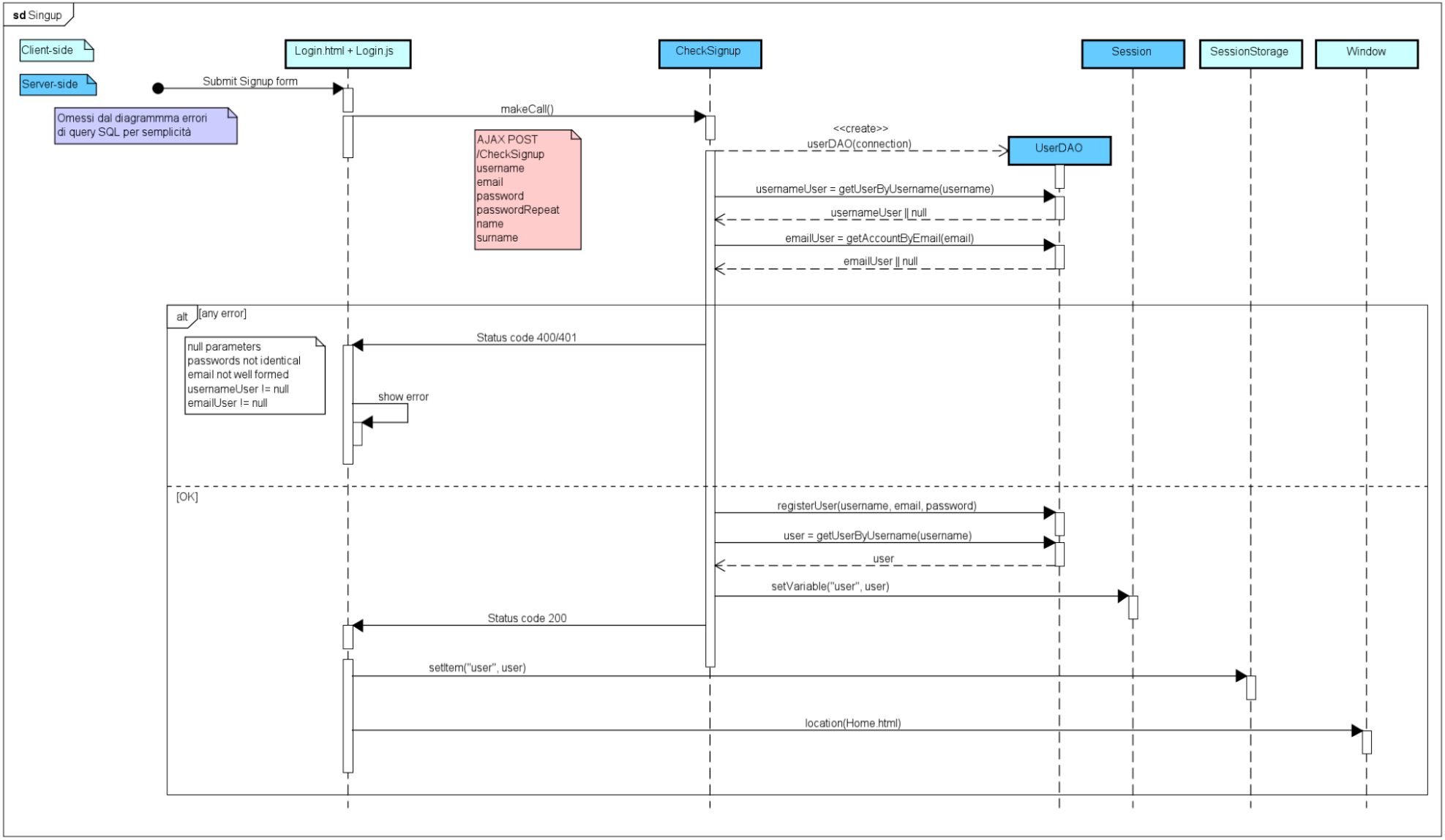
Home

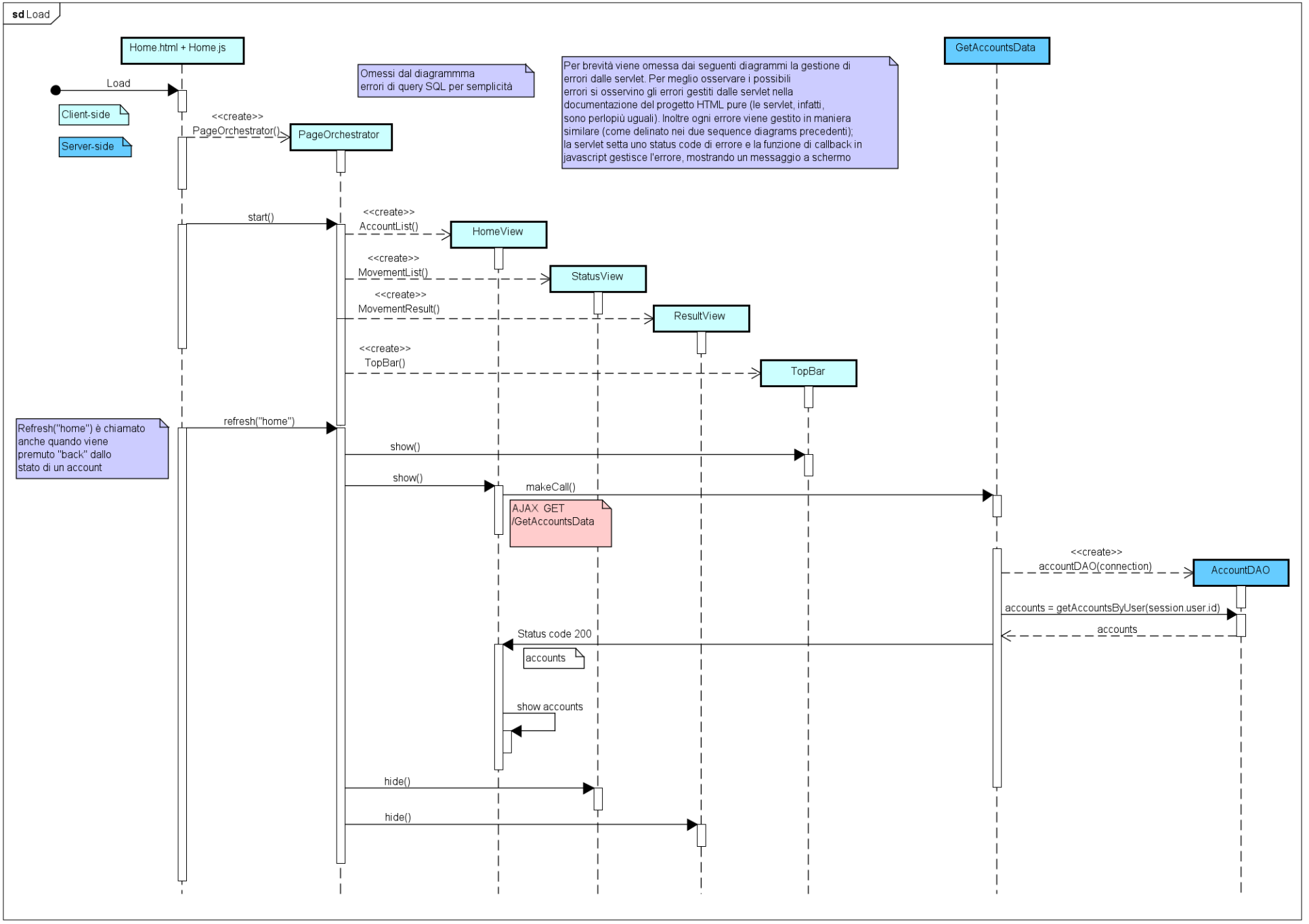
(per “vista” si intende un insieme di componenti che compongono una visualizzazione specifica, non una pagina a sé stante, in quanto Home è l’unica pagina come da specifiche)

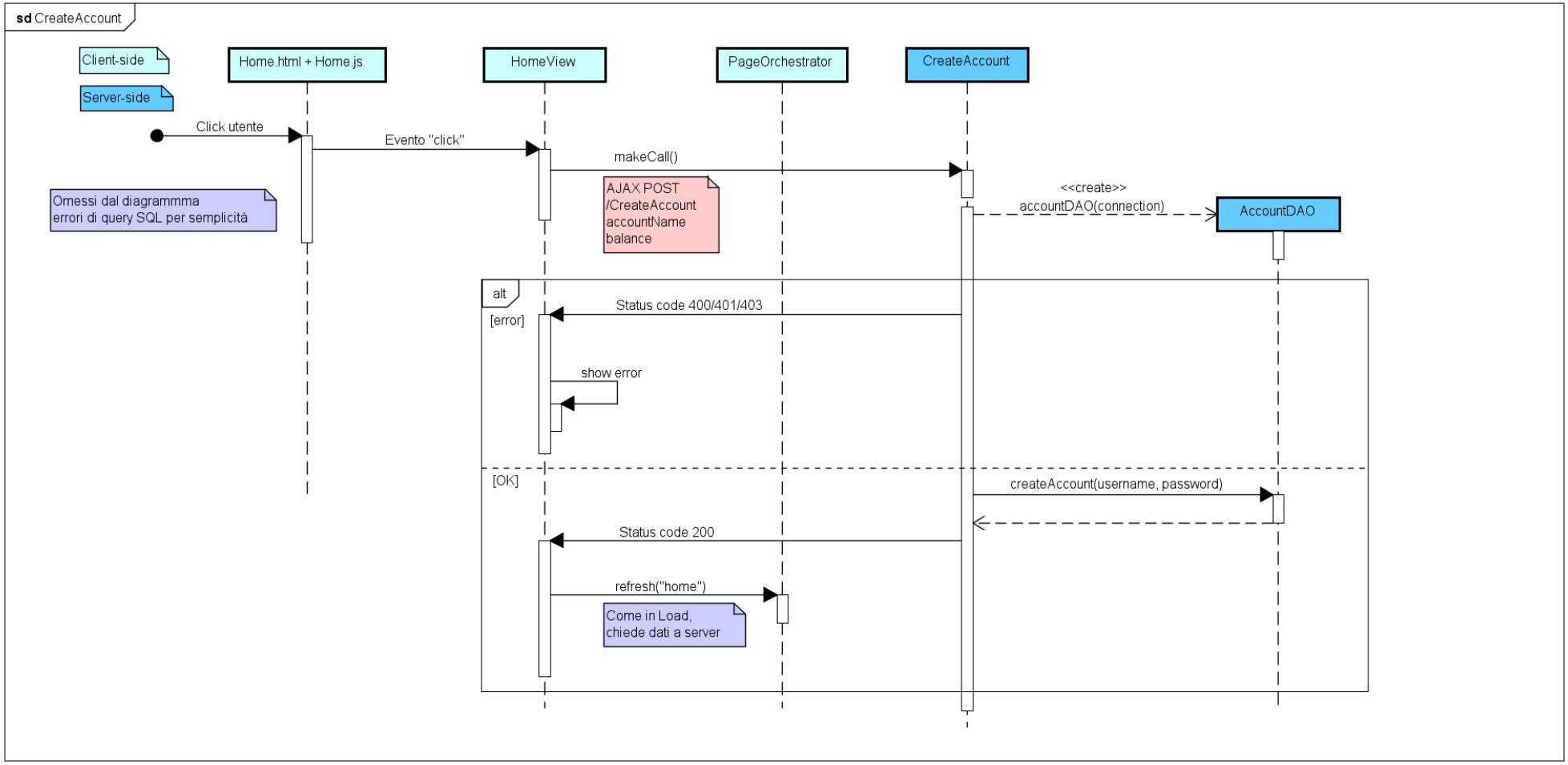
- Page Orchestrator
  - start(): inizializza tutti i componenti della Home passandogli gli elementi HTML su cui agiscono
  - refresh(“view”): aggiorna la vista a schermo mostrando la “view” richiesta. In altre parole, nasconde tutte le viste (hide) e mostra (show) quella richiesta
- TopBar
  - showBack(previous): mostra il pulsante di “back” per tornare alla “vista” precedente
  - hideBack(): nasconde il pulsante di back
- HomeView
  - show(): richiede al server la lista account e la mostra all’interno della “vista” home
  - hide(): nasconde gli elementi relativi alla “vista” home
- StatusView
  - show(): richiede al server le liste dei movimenti e le mostra nella “vista” dello stato dell’account account. Richiede inoltre la lista dei contatti dell’utente e la salva in addressBook
  - hide(): nasconde gli elementi relativi alla “vista” dello stato
- ResultView
  - show(): mostra l’errore o il successo del trasferimento, a seconda del valore salvato. Se il movimento è avvenuto con successo e l’account a cui si è trasferito denaro non è presente nell’addressBook (salvato in precedenza) allora mostra anche un prompt per inserire l’utente tra i propri contatti
  - hide(): nasconde gli elementi relativi alla “vista” del risultato

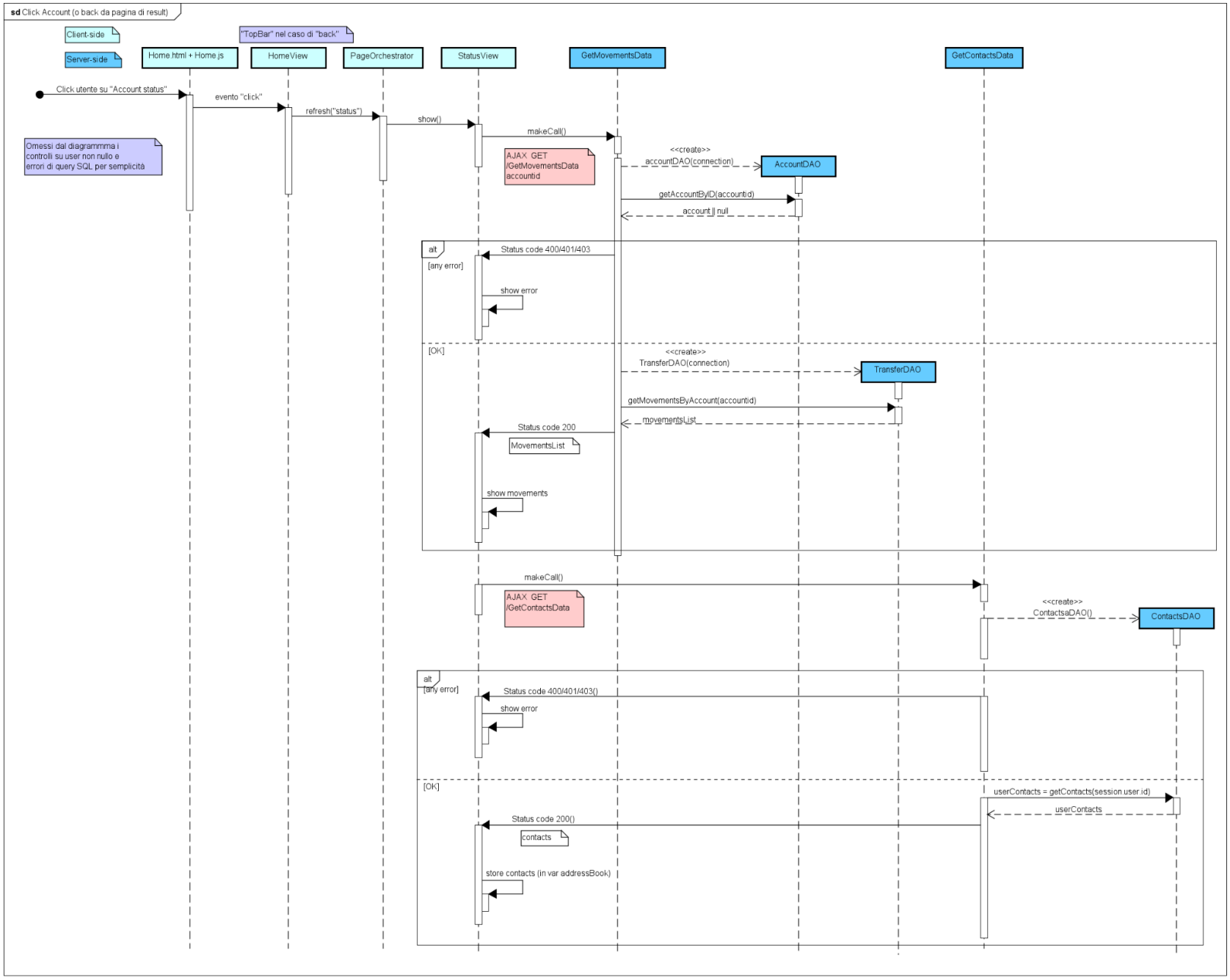
Sequence diagrams

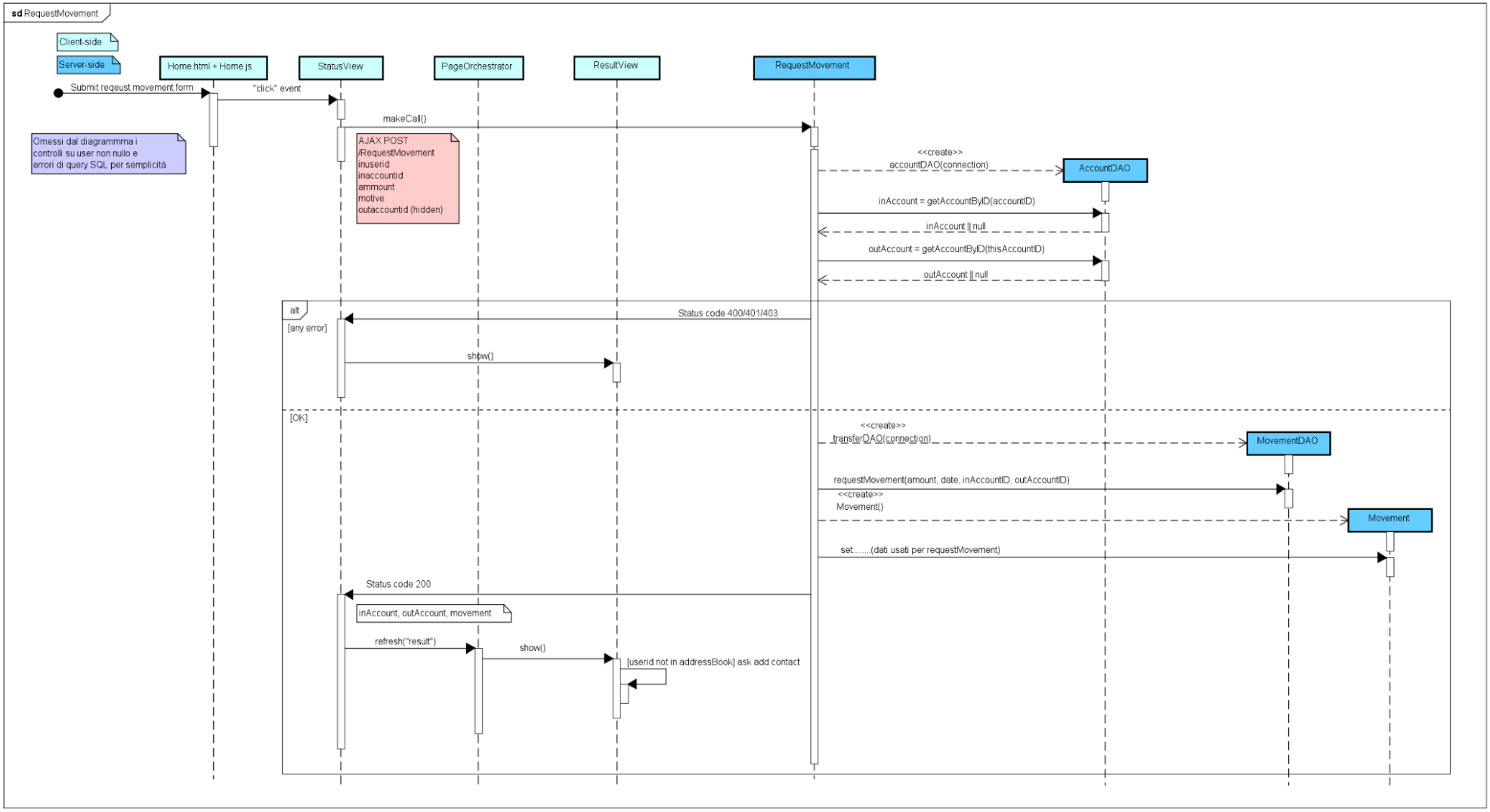


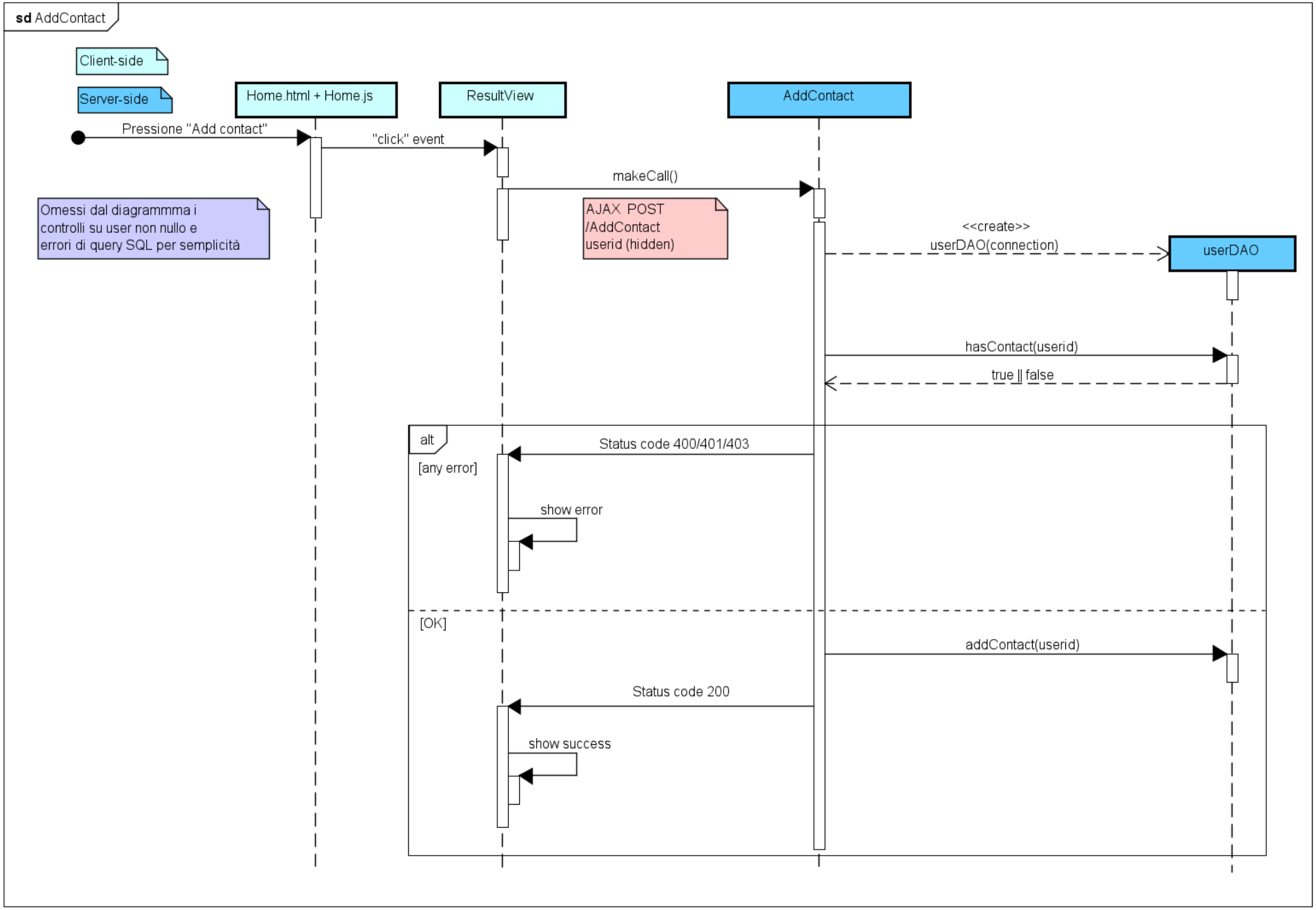

sd Singup
Client-side
Server-side
Login.html + Login.js
CheckSignup
Session
SessionStorage
Window
Submit Signup form
Omessi dal diagrammma errori
di query SQL per semplicità
makeCall()
AJAX POST
/CheckSignup
username
email
password
passwordRepeat
name
surname
<<create>>
userDAO(connection)
UserDAO
usernameUser = getUserByUsername(username)
usernameUser || null
emailUser = getAccountByEmail(email)
emailUser || null
alt
[any error]
null parameters
passwords not identical
email not well formed
usernameUser != null
emailUser != null
Status code 400/401
show error
[OK]
registerUser(username, email, password)
user = getUserByUsername(username)
user
setVariable("user", user)
Status code 200
setItem("user", user)
location(Home.html)

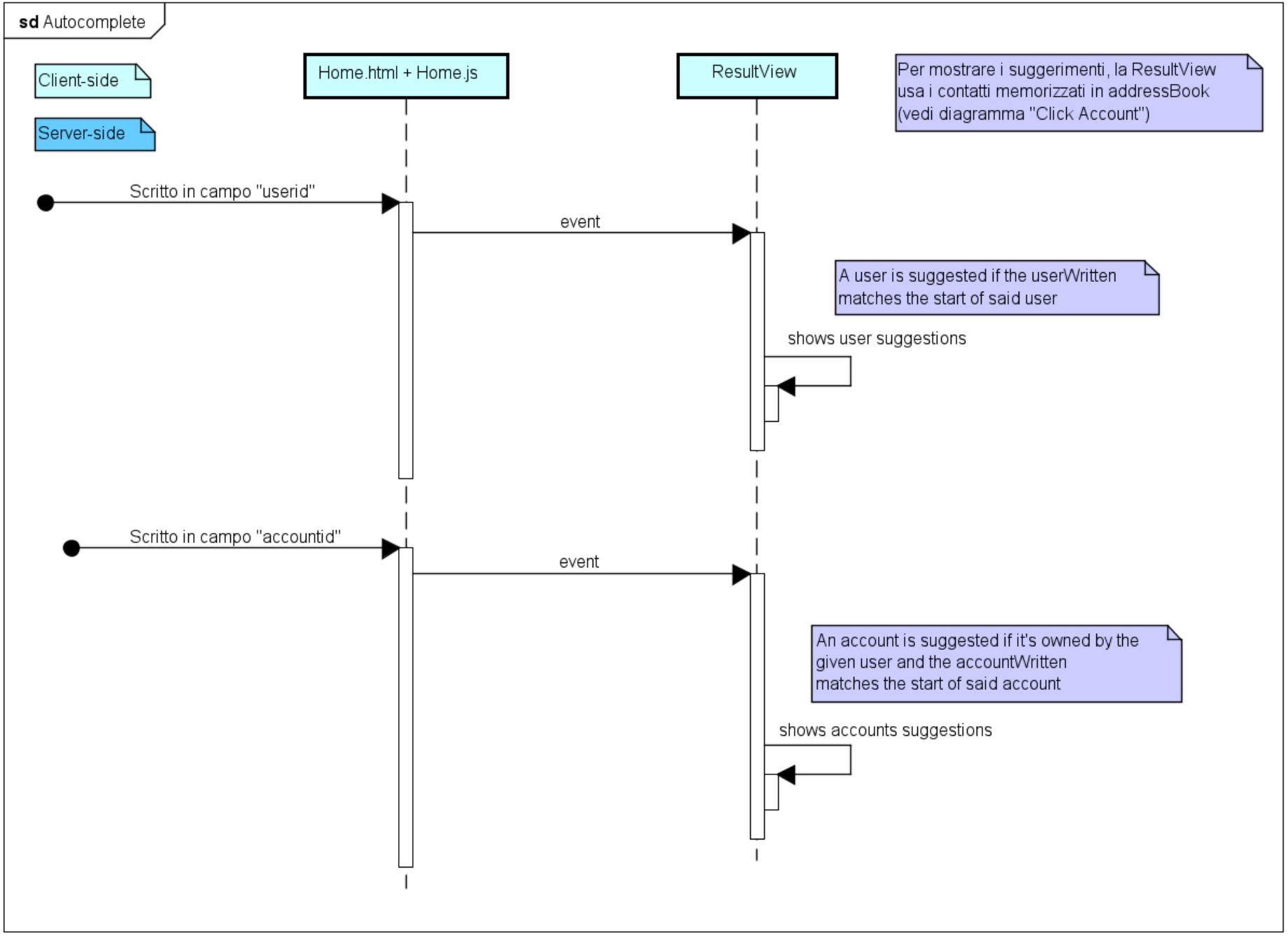

sd Load
Home.html + Home.js
GetAccountsData
Load
Client-side
Server-side
<<create>>
PageOrchestrator()
PageOrchestrator
Omessi dal diagrammma
errori di query SQL per semplicità
Per brevità viene omessa dai seguenti diagrammi la gestione di
errori dalle servlet. Per meglio osservare i possibili
errori si osservino gli errori gestiti dalle servlet nella
documentazione del progetto HTML pure (le servlet, infatti,
sono perlopiù uguali). Inoltre ogni errore viene gestito in maniera
similare (come delinato nei due sequence diagrams precedenti);
la servlet setta uno status code di errore e la funzione di callback in
javascript gestisce l'errore, mostrando un messaggio a schermo
start()
<<create>>
AccountList()
HomeView
<<create>>
MovementList()
StatusView
<<create>>
MovementResult()
ResultView
<<create>>
TopBar()
TopBar
Refresh("home") è chiamato
anche quando viene
premuto "back" dallo
stato di un account
refresh("home")
show()
show()
makeCall()
AJAX GET
/GetAccountsData
<<create>>
accountDAO(connection)
AccountDAO
accounts = getAccountsByUser(session.user.id)
accounts
Status code 200
accounts
show accounts
hide()
hide()

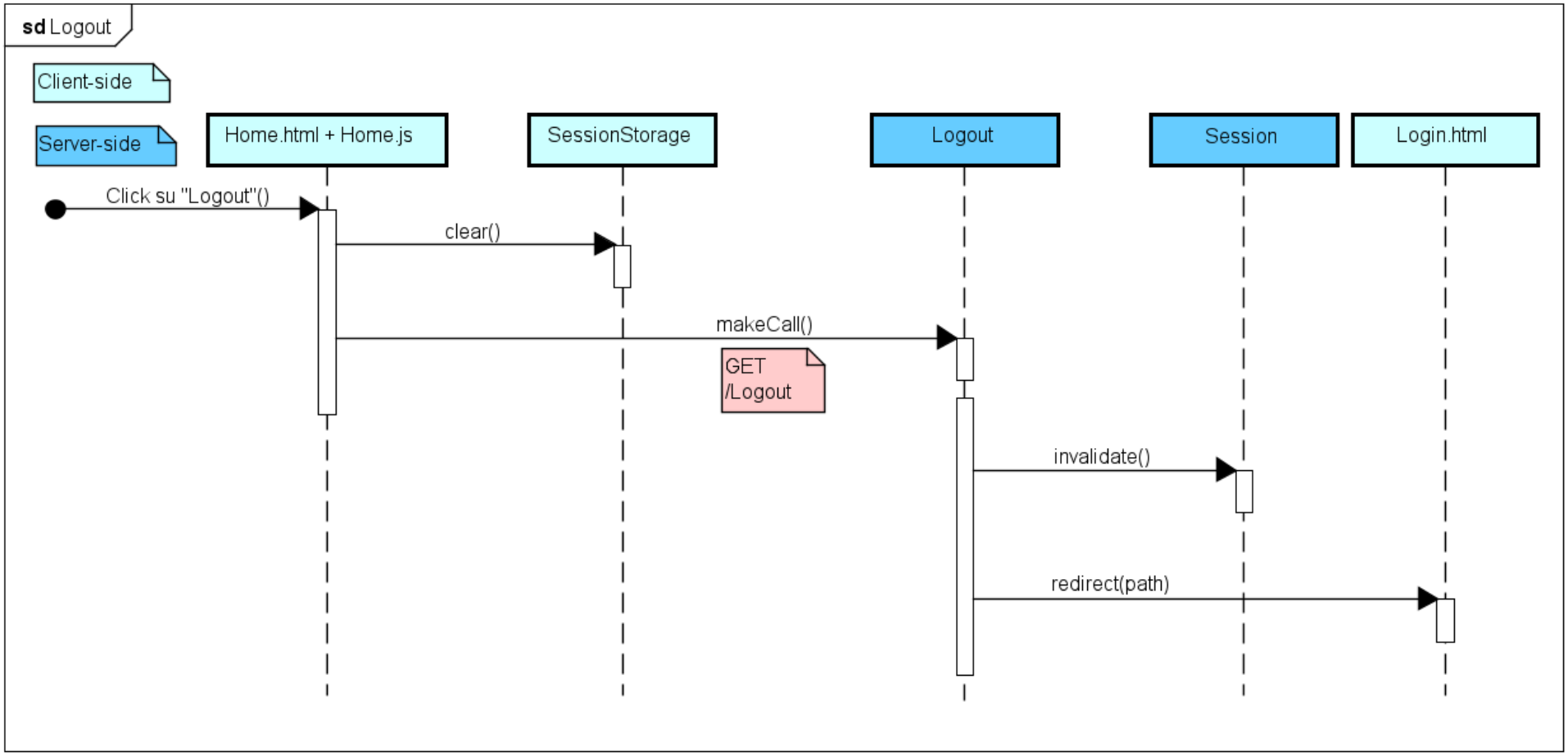

sd CreateAccount
Client-side
Server-side
Home.html + Home.js
HomeView
PageOrchestrator
CreateAccount
AccountDAO
Click utente
Evento "click"
makeCall()
AJAX POST
/CreateAccount
accountName
balance
Omessi dal diagrammma
errori di query SQL per semplicità
<<create>>
accountDAO(connection)
alt
[error]
Status code 400/401/403
show error
[OK]
createAccount(username, password)
Status code 200
refresh("home")
Come in Load,
chiede dati a server


sd Click Account (o back da pagina di result)
Client-side
Server-side
"TopBar" nel caso di "back"
Home.html + Home.js
HomeView
PageOrchestrator
StatusView
GetMovementsData
GetContactsData
Click utente su "Account status"
evento "click"
refresh("status")
show()
makeCall()
Omessi dal diagrammma i controlli su user non nullo e errori di query SQL per semplicità
AJAX GET /GetMovementsData accountid
<<create>>
accountDAO(connection)
AccountDAO
getAccountByID(accountid)
account || null
alt
[any error]
Status code 400/401/403
show error
[OK]
<<create>>
TransferDAO(connection)
TransferDAO
getMovementsByAccount(accountid)
movementsList
Status code 200
MovementsList
show movements
makeCall()
AJAX GET /GetContactsData
<<create>>
ContactsaDAO()
ContactsDAO
alt
[any error]
Status code 400/401/403()
show error
[OK]
userContacts = getContacts(session.user.id)
userContacts
Status code 200()
contacts
store contacts (in var addressBook)


sd RequestMovement
Client-side
Server-side
Home.html + Home.js
StatusView
PageOrchestrator
ResultView
RequestMovement
Submit request movement form
"click" event
makeCall()
Omessi dal diagrammma i controlli su user non nullo e errori di query SQL per semplicità
AJAX POST
/RequestMovement
inuserid
inaccountid
ammount
motive
outaccountid (hidden)
<<create>>
accountDAO(connection)
AccountDAO
inAccount = getAccountByID(accountID)
inAccount || null
outAccount = getAccountByID(thisAccountID)
outAccount || null
alt
[any error]
Status code 400/401/403
show()
[OK]
<<create>>
transferDAO(connection)
MovementDAO
requestMovement(amount, date, inAccountID, outAccountID)
<<create>>
Movement()
Movement
set.....(dati usati per requestMovement)
Status code 200
inAccount, outAccount, movement
refresh("result")
show()
[userid not in addressBook] ask add contact


sd AddContact
Client-side
Server-side
Home.html + Home.js
ResultView
AddContact
Pressione "Add contact"
"click" event
makeCall()
Omessi dal diagrammma i
controlli su user non nullo e
errori di query SQL per semplicità
AJAX POST
/AddContact
userid (hidden)
<<create>>
userDAO(connection)
userDAO
hasContact(userid)
true || false
alt
[any error]
Status code 400/401/403
show error
[OK]
addContact(userid)
Status code 200
show success


sd Autocomplete
Client-side
Server-side
Home.html + Home.js
ResultView
Per mostrare i suggerimenti, la ResultView usa i contatti memorizzati in addressBook (vedi diagramma "Click Account")
Scritto in campo "userid"
event
A user is suggested if the userWritten matches the start of said user
shows user suggestions
Scritto in campo "accountid"
event
An account is suggested if it's owned by the given user and the accountWritten matches the start of said account
shows accounts suggestions


sd Logout
Client-side
Server-side
Home.html + Home.js
SessionStorage
Logout
Session
Login.html
Click su "Logout"()
clear()
makeCall()
GET
/Logout
invalidate()
redirect(path)