Esercizio 4 versione RIA

Luca Lain - Sergio Lupo

Esercizio 4: trasferimento denaro

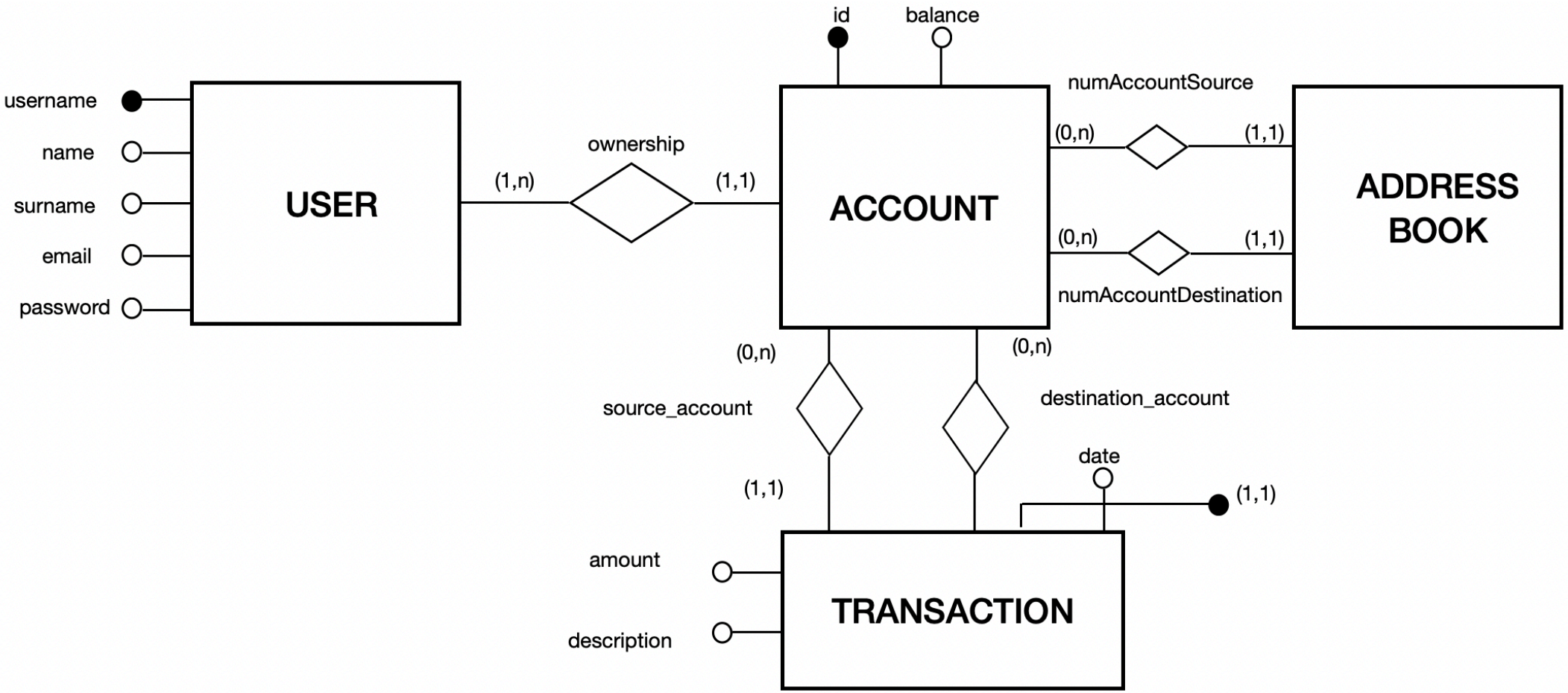
Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- 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

(Non sono state rievidenziate le entità, relazioni e attributi già evidenziati nella versione pure HTML)

Database Design



Database Schema(1/3)

```
CREATE TABLE `user` (
  `username` varchar(255) NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  `email` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`username`)
) ENGINE=InnoDB;
CREATE TABLE `account` (
  `id` int NOT NULL AUTO_INCREMENT,
  `balance` float NOT NULL,
  `idUser` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idUser` (`idUser`),
  CONSTRAINT `idUser` FOREIGN KEY (`idUser`) REFERENCES `user` (`username`) ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Database Schema(2/3)

```
CREATE TABLE `transaction` (
  `sourceAccount` int NOT NULL,
  `destinationAccount` int NOT NULL,
  `id` int NOT NULL AUTO_INCREMENT,
  `date` date NOT NULL,
  `amount` float NOT NULL,
  `description` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `sourceAccount` (`sourceAccount`),
  KEY `destinationAccount` (`destinationAccount`),
  CONSTRAINT `destinationAccount` FOREIGN KEY (`destinationAccount`) REFERENCES `account` (`id`) ON UPDATE CASCADE,
  CONSTRAINT `sourceAccount` FOREIGN KEY (`sourceAccount`) REFERENCES `account` (`id`) ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Database Schema(3/3)

```
CREATE TABLE `address_book` (
  `numAccountSource` int NOT NULL,
  `numAccountDestination` int NOT NULL,
  PRIMARY KEY (`numAccountSource`,`numAccountDestination`),
  KEY `numAccountSource_idx` (`numAccountSource`),
  KEY `numAccountDestination_idx` (`numAccountDestination`),
  CONSTRAINT `numAccountDestination` FOREIGN KEY (`numAccountDestination`) REFERENCES `account` (`id`),
  CONSTRAINT `numAccountSource` FOREIGN KEY (`numAccountSource`) REFERENCES `account` (`id`)
) ENGINE=InnoDB;
```

Analisi Requisiti Applicazione

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

- Ogni utente viene identificato dal suo codice utente.
- Per credenziali di login si intendono username e password.
- Se l'utente è loggato, verrà reindirizzato automaticamente alla pagina protetta se tenta di accedere alla pagina di registrazione. Viceversa se l'utente non è loggato, verrà reindirizzato alla pagina di login.
- Se l'utente è loggato, è possibile fare logout e tornare quindi alla pagina di login.
- Non è possibile fare trasferimenti con conto origine uguale al conto destinazione.
- L'avviso di conferma trasferimento è realizzato mediante un messaggio nella pagina che ospita l'applicazione.
- Dalla pagina home, è anche possibile creare un nuovo conto e aggiungere soldi ad un conto già esistente.
- Durante l'inserimento del codice del destinatario, vengono presentati come suggerimenti in un menu a tendina l'elenco dei destinatari in rubrica. Selezionato il destinatario, vengono presentati come suggerimenti in un menu a tendina i suoi conti in rubrica. Se il conto del destinatario non è in rubrica l'utente ha la possibilità di aggiungerlo.

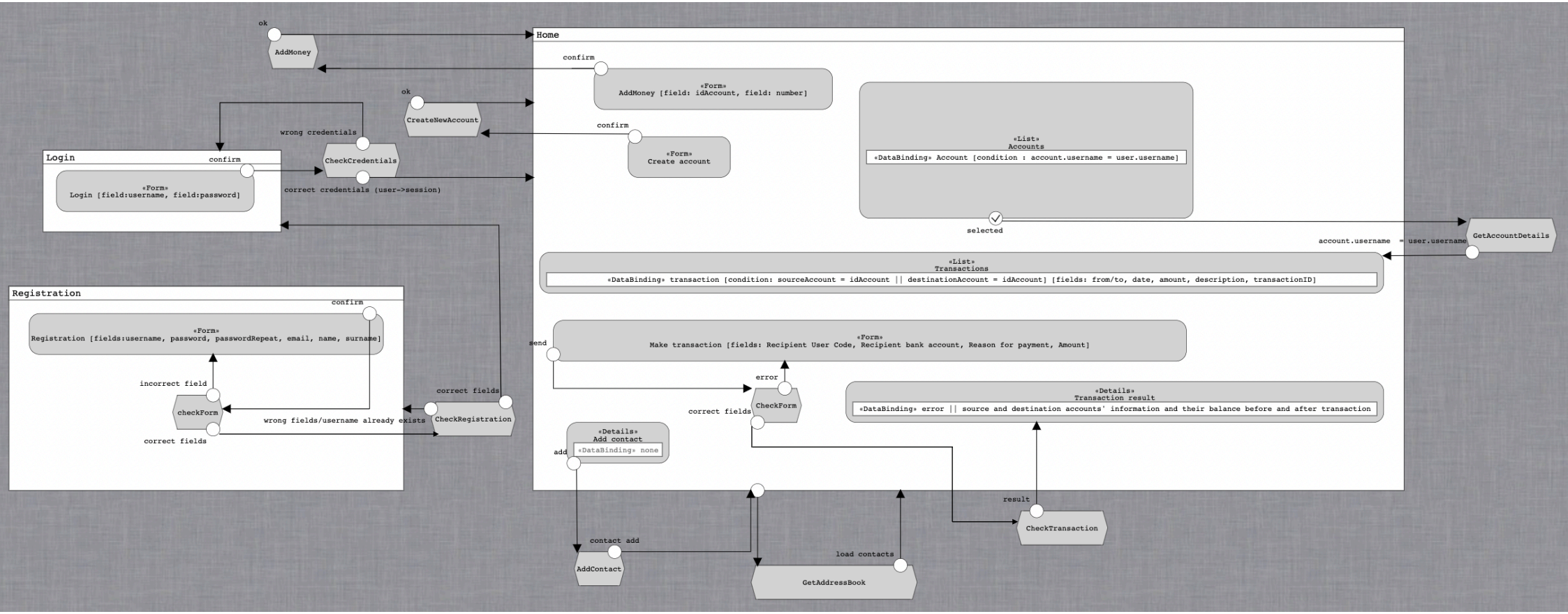
Server-Side Components

- Model object (Beans)
  - User
  - Account
  - Transaction
  - AddressBook
- Data Access Object (DAO)
  - UserDAO
    - ✦ insertUser(User user,String password) : boolean
    - ✦ checkLogin(String usr, String pwd) : User
  - AccountDAO
    - ✦ getTransactions(int idUser) : ArrayList<Transaction>
  - TransactionDAO
    - ✦ getAccountsFromIdUser(String idUser) : ArrayList<Account>
    - ✦ getAccountFromId(int id) : Account
    - ✦ updateAccount(Account source,Account destination,Transaction transaction) : boolean
    - ✦ insertNewAccount(User user) : boolean
    - ✦ addMoney(int id, float num) : boolean
  - AddressBookDAO
    - ✦ insertNewContact(int source, int destination) : boolean
    - ✦ existsContactEntry(int idCurrentAccount, int idDestinationAccount) : boolean
- Filters
  - UserNotLogged
  - NoCacher
  - LoginChecker
- Controllers (servlets) [access right]
  - Logout [loggedUser]
  - GetAddressBook [loggedUser]
  - GetAccounts [loggedUser]
  - GetAccountDetails [loggedUser]
  - CreateNewAccount [loggedUser]
  - CheckTransaction [loggedUser]
  - CheckRegistration [notLoggedUser]
  - CheckCredentials [all]
  - AddMoney [loggedUser]
  - AddContact [loggedUser]
- Views (Templates)
  - Registration.html
  - index.html (the login page)
  - Home.html

Client-Side Components

- Login (index)
  - Login form
    - ✦ manage submit and errors
  - Registration form
    - ✦ manage submit and errors
- Home
  - PageOrchestrator
    - ✦ start() : crea i componenti della pagina
    - ✦ refresh(excludeContacts): carica i contenuti dei componenti e la loro visualizzazione
  - AccountList
    - ✦ show() : richiede al server i dati dell'elenco degli account
    - ✦ update() : riceve i dati degli account e aggiorna la lista
    - ✦ autoClick() : seleziona il primo account della lista e mostra le sue transazioni
  - AccountDetail
    - ✦ show() : richiede al server le transazioni
    - ✦ update() : riceve le transazioni e aggiorna i dettagli
  - Transaction
    - ✦ showTransaction() : mostra i dettagli della transazione
    - ✦ hideTransaction() : cancella i dettagli della transazione
    - ✦ initialize() : inizializza i due bottoni per aggiungere soldi e per fare la transazione
  - AddressBook
    - ✦ load(): richiede al server i dati della rubrica dell'account selezionato
    - ✦ autoCompleteDestinationUsername(): suggerisce all'utente il completamento dell'username del destinatario
    - ✦ autoCompleteDestinationAccount(): suggerisce all'utente il completamento dell'account dell'username destinatario
    - ✦ showButton(): mostra il bottone per aggiungere un nuovo contatto
    - ✦ hideButton(): nasconde il bottone per aggiungere un nuovo contatto

Design Applicativo (IFML)



Events & Actions

Evento client side	Azione client side	Evento server side	Azione server side
index.html -> form -> submit	Controllo dati	POST username password	Controllo credenziali
Register.html -> form -> submit	Controllo uguaglianza password e controllo del campo email	POST username password passwordRepeat email name surname	Controllo campi e inserimento utente
Home.html -> load	Aggiorna view con lista account	POST nessun parametro	Estrazione lista account
Home.html -> elenco account -> seleziona account	Aggiorna view e mostra transazioni dell'account selezionato	GET accountid	Estrazione lista transazioni e lista contatti dell'account selezionato
Home.html -> form aggiungi soldi -> send	Controllo dati	POST accountid number (soldi da aggiungere)	Controllo accountid e aggiunta di soldi
Home.html -> creazione nuovo conto		POST nessun parametro	Aggiunta account
Home.html -> form transazione ->send	Controllo dati	POST idUser idAccount description amount idThis	Controllo transazione e invio risultato
Home.html -> transazione completata	Mostra bottone add_contact se il conto destinatario non è salvato in rubrica		
Home.html -> aggiungi contatto	Ajax post	POST sourceAccount destinationUser destinationId	Aggiungi contatto
Home.html -> inserimento dell'input nella form della transazione	Mostra suggerimenti		

Controller & Event Handlers

Evento client side	Controllore client side	Evento server side	Controllore server side
index -> form -> submit	Function makeCall	POST username password	CheckCredentials
Registration -> form -> submit	Function makeCall	POST username password passwordRepeat email name surname	CheckRegistration
Home -> load	Function PageOrchestrator	POST nessun parametro	GetAccounts
Home -> elenco account -> seleziona account	Function AccountDetail.show	GET accountid	GetAccountDetails
Home -> form aggiungi soldi -> send	Function makeCall	POST accountid number (soldi da aggiungere)	AddMoney
Home -> creazione nuovo conto	Function makeCall	POST nessun parametro	CreateNewAccount
Home -> form transazione ->send	Function makeCall	POST idUser idAccount description amount idThis	CheckTransaction
Home -> transazione completata	Function AccountList.show		
Home -> aggiungi contatto	Function makeCall	POST sourceAccount destinationUser destinationId	AddContact
Home -> inserimento dell'input nella form della transazione	Function addressBook.autocompleteDestinationUsername addressBook.autocompleteDestinationAccount		

Sequence diagrams

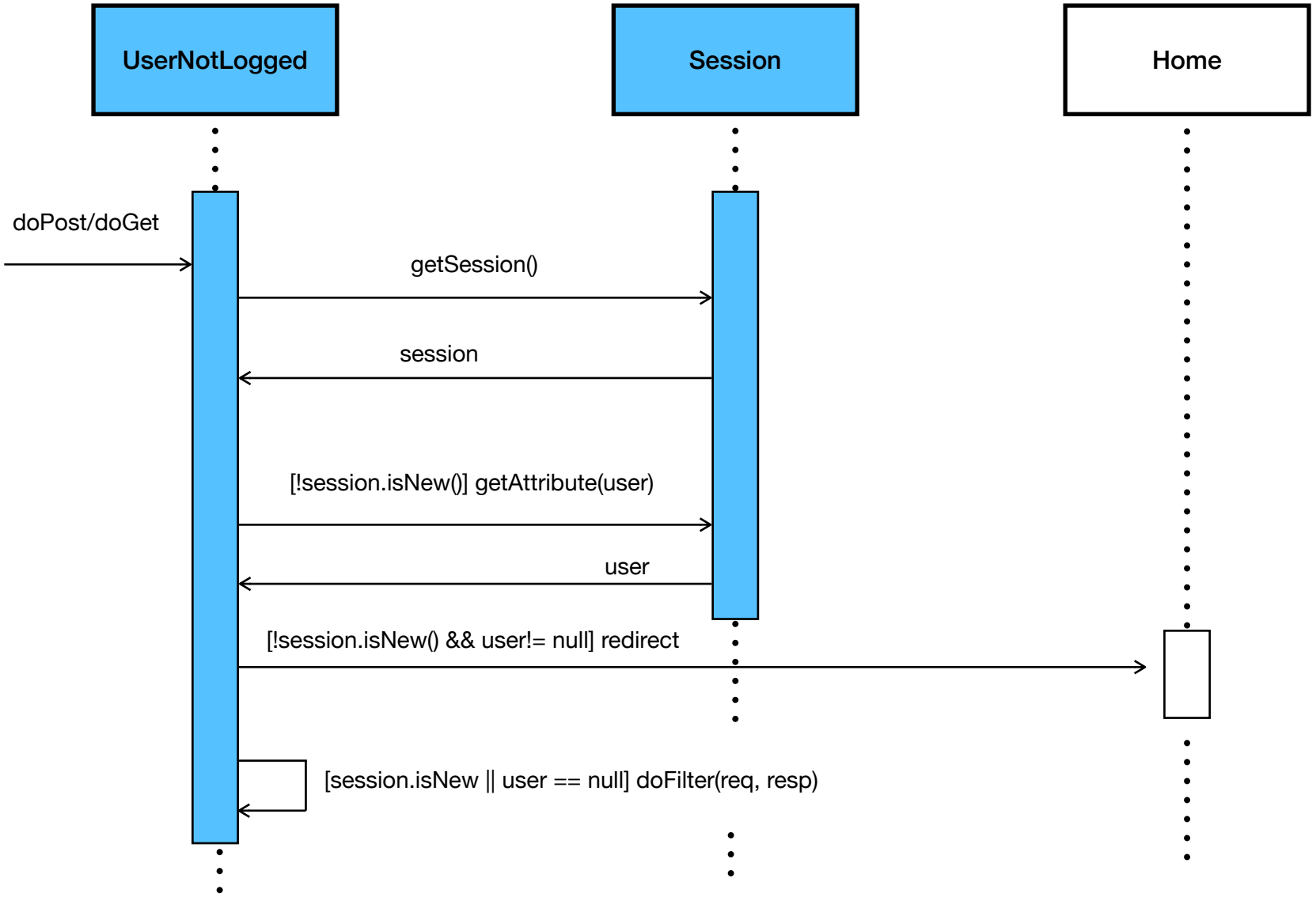
Di seguito sono riportati i sequence diagrams che rappresentano gli eventi principali dell'applicazione Web. Alcuni dettagli minori sono stati tralasciati, ovvero gli errori interni del server, gli errori di accesso al database e l'inserimento di parametri nulli. Inoltre sono state semplificate alcune interazioni lato-client (ad esempio alcune che non implicano chiamate AJAX).

Per semplicità i controlli dei filtri sono rappresentati nei primi schemi e sono omessi in quelli successivi.

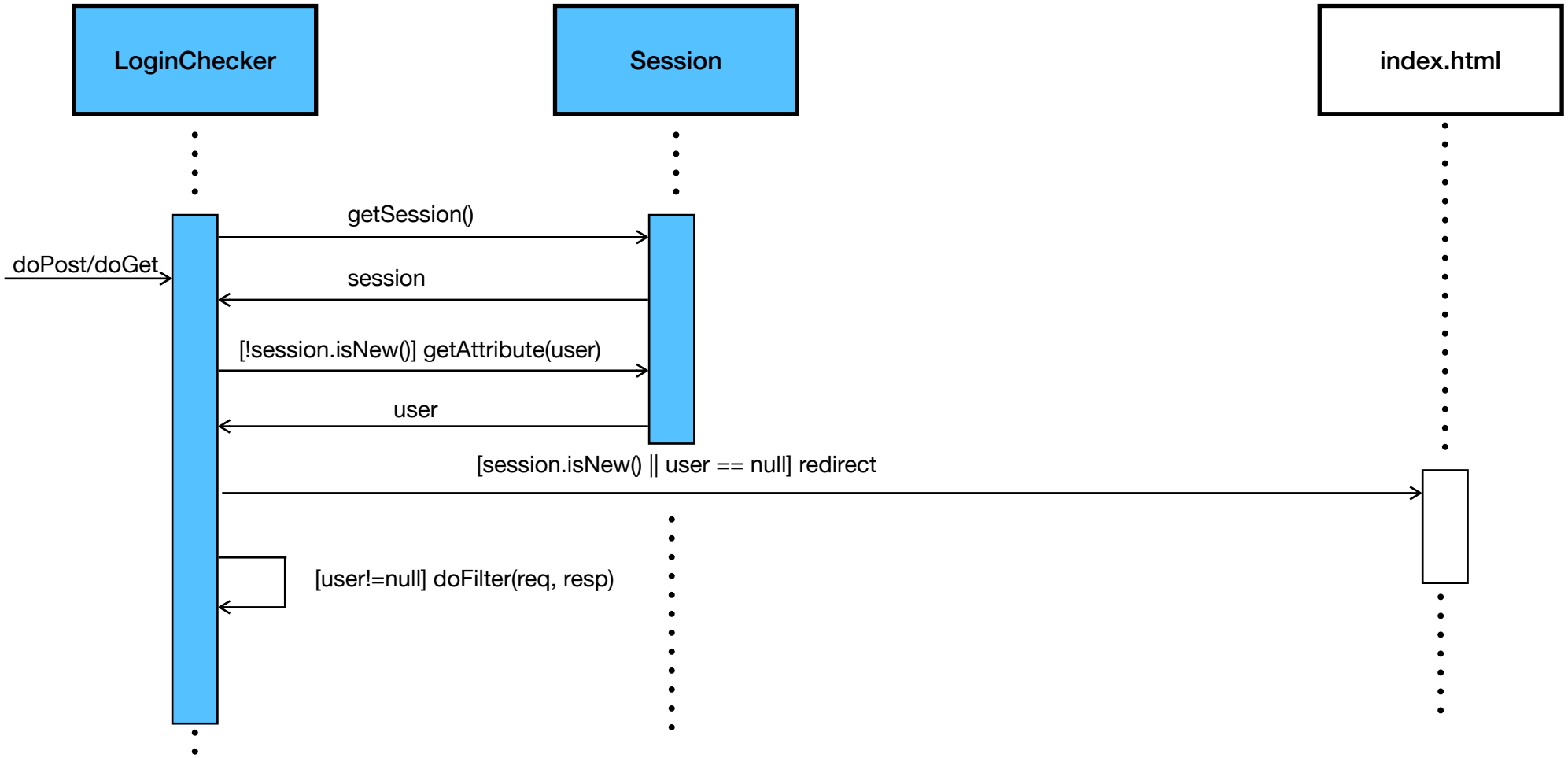
I componenti lato server sono indicati in azzurro, mentre i componenti lato client in bianco.

Per ogni sequence diagram è riportato in alto il nome dell'evento.

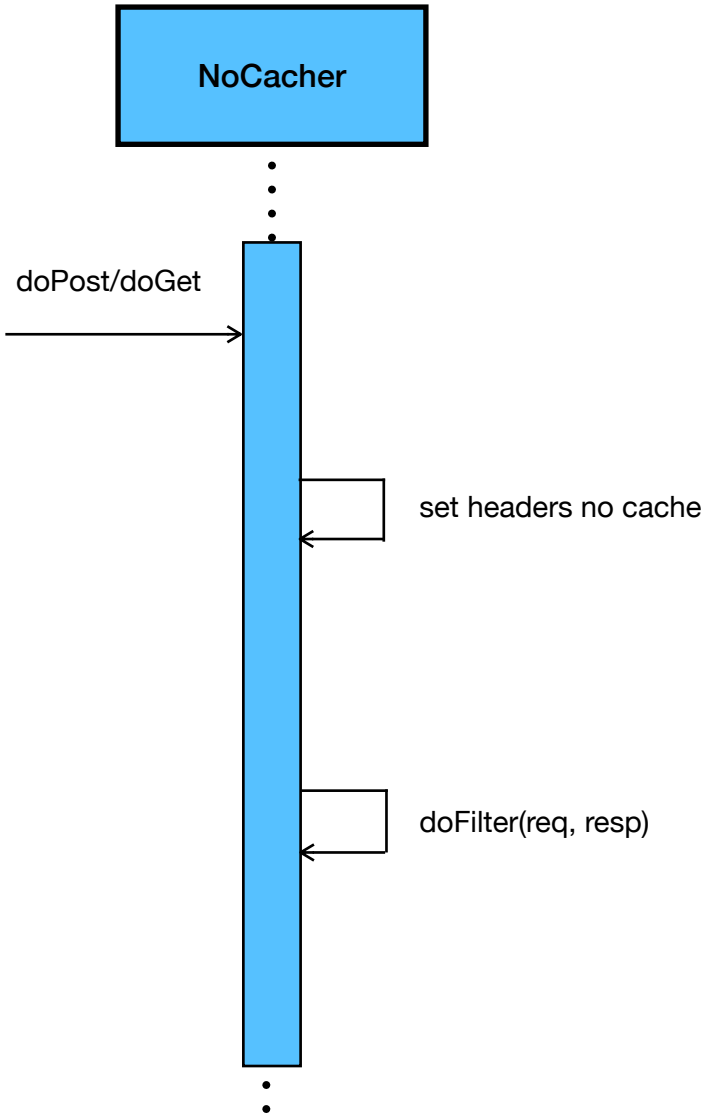
UserNotLogged (Filter)



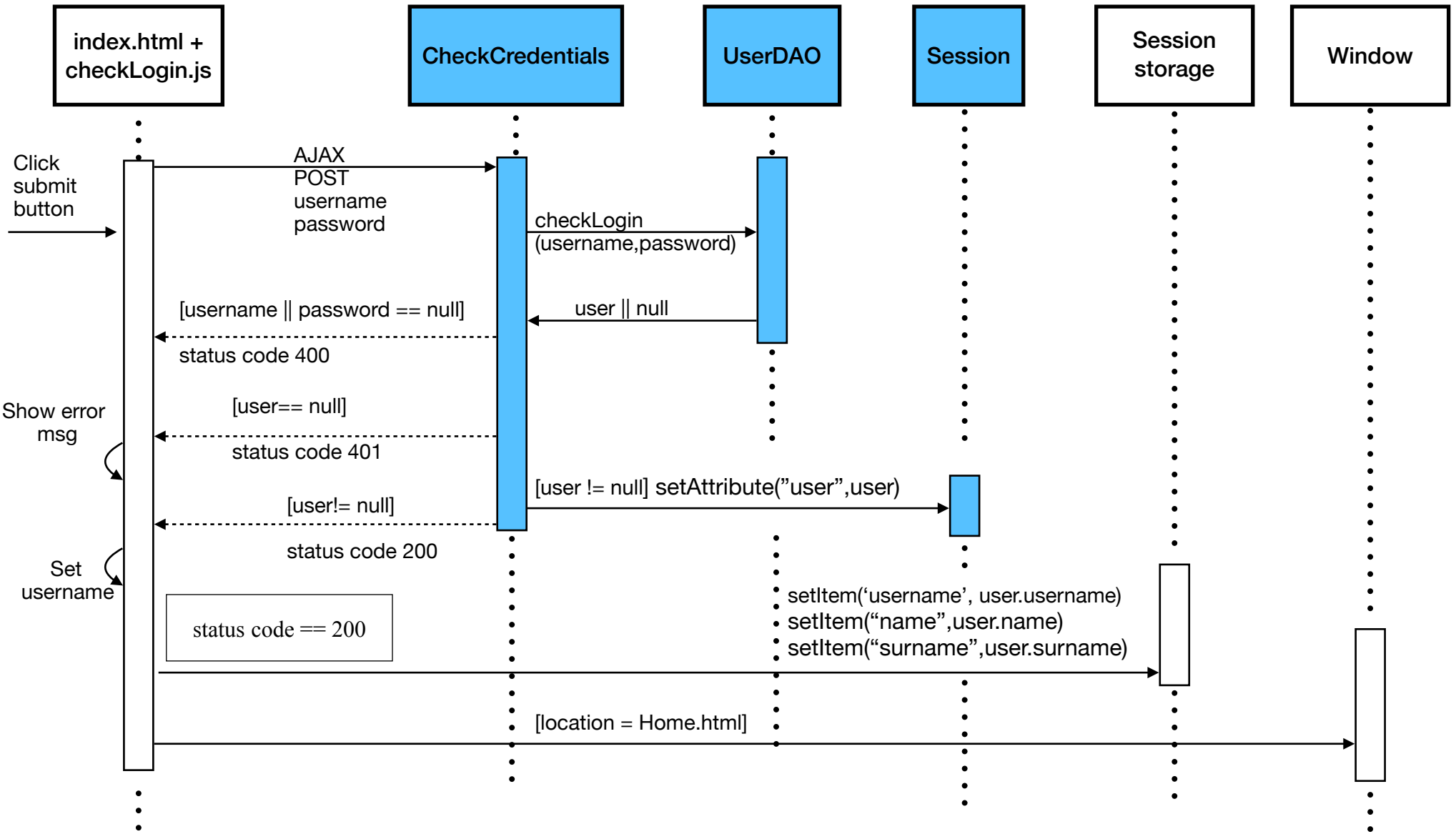
LoginChecker (Filter)



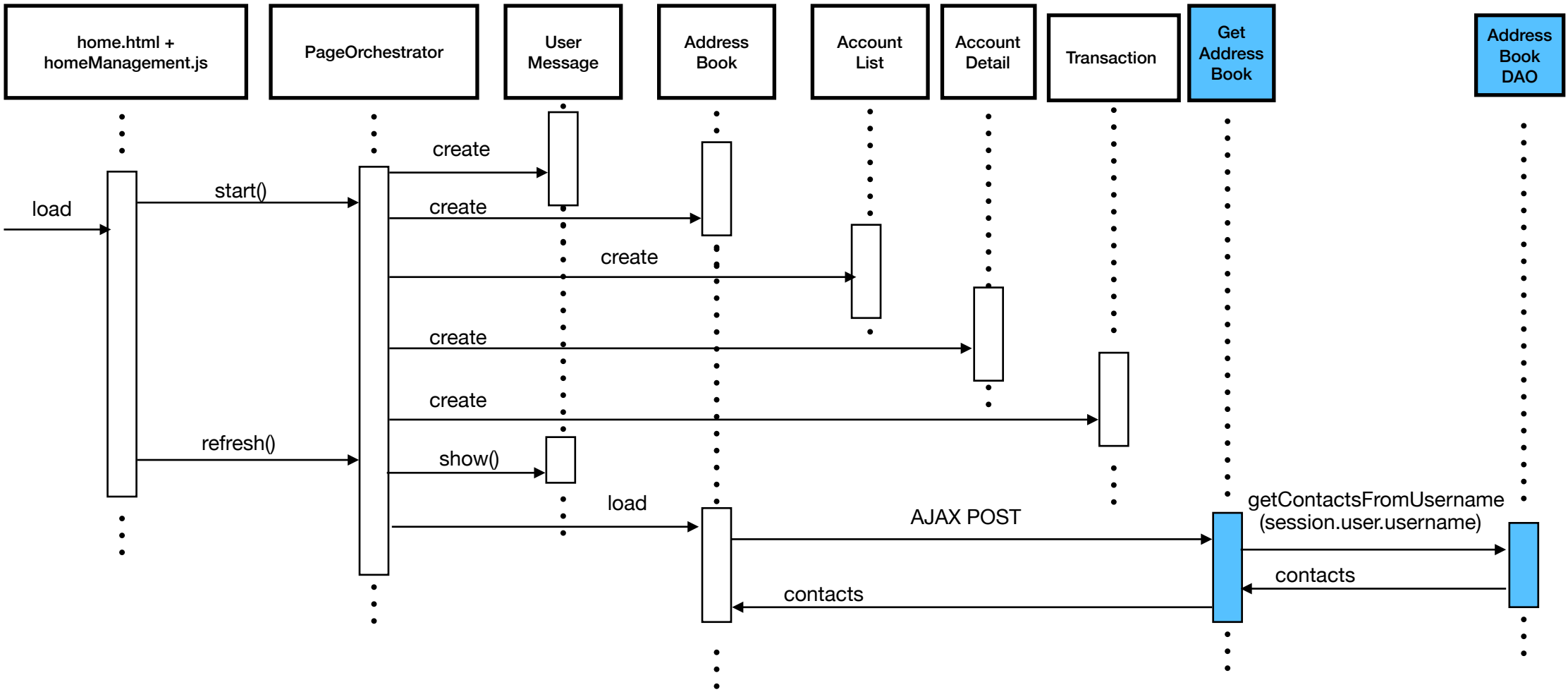
NoCacher (Filter)



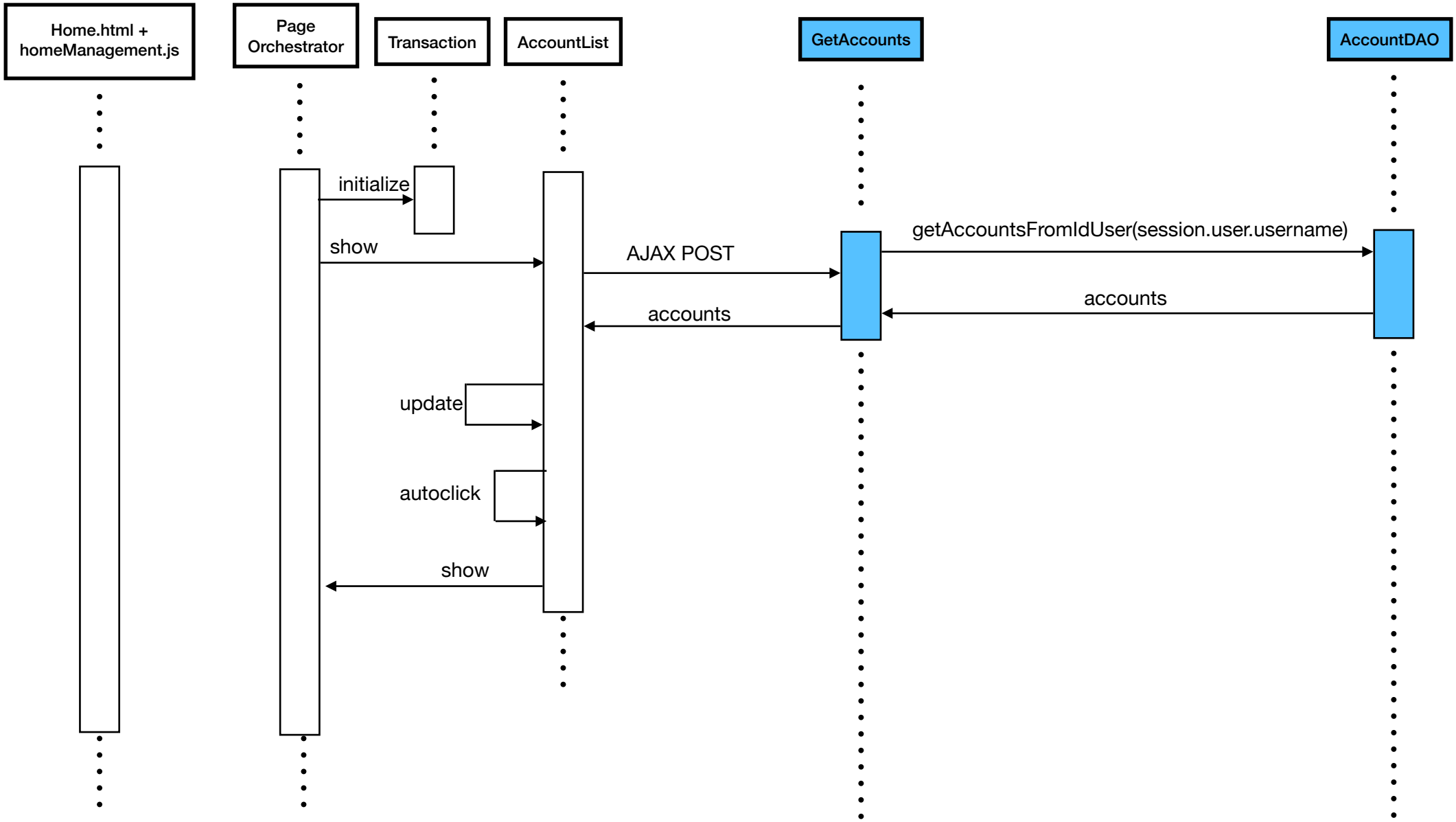
Login


index.html + checkLogin.js
CheckCredentials
UserDAO
Session
Session storage
Window
Click submit button
AJAX POST username password
checkLogin (username,password)
user || null
[username || password == null]
status code 400
Show error msg
[user== null]
status code 401
[user != null] setAttribute(”user”,user)
[user!= null]
status code 200
Set username
status code == 200
setItem(‘username’, user.username)
setItem(“name”,user.name)
setItem(“surname”,user.surname)
[location = Home.html]

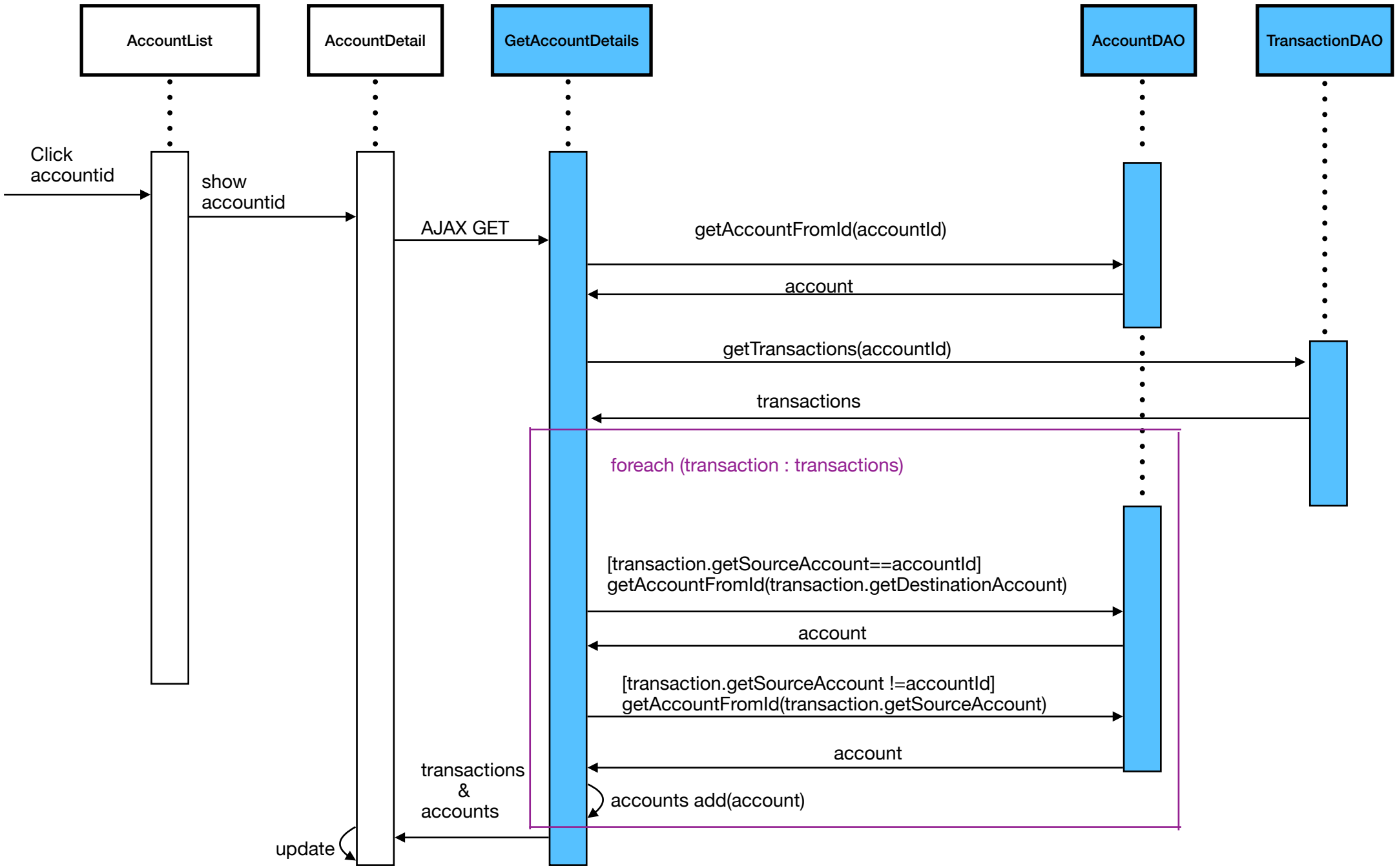
Caricamento Home



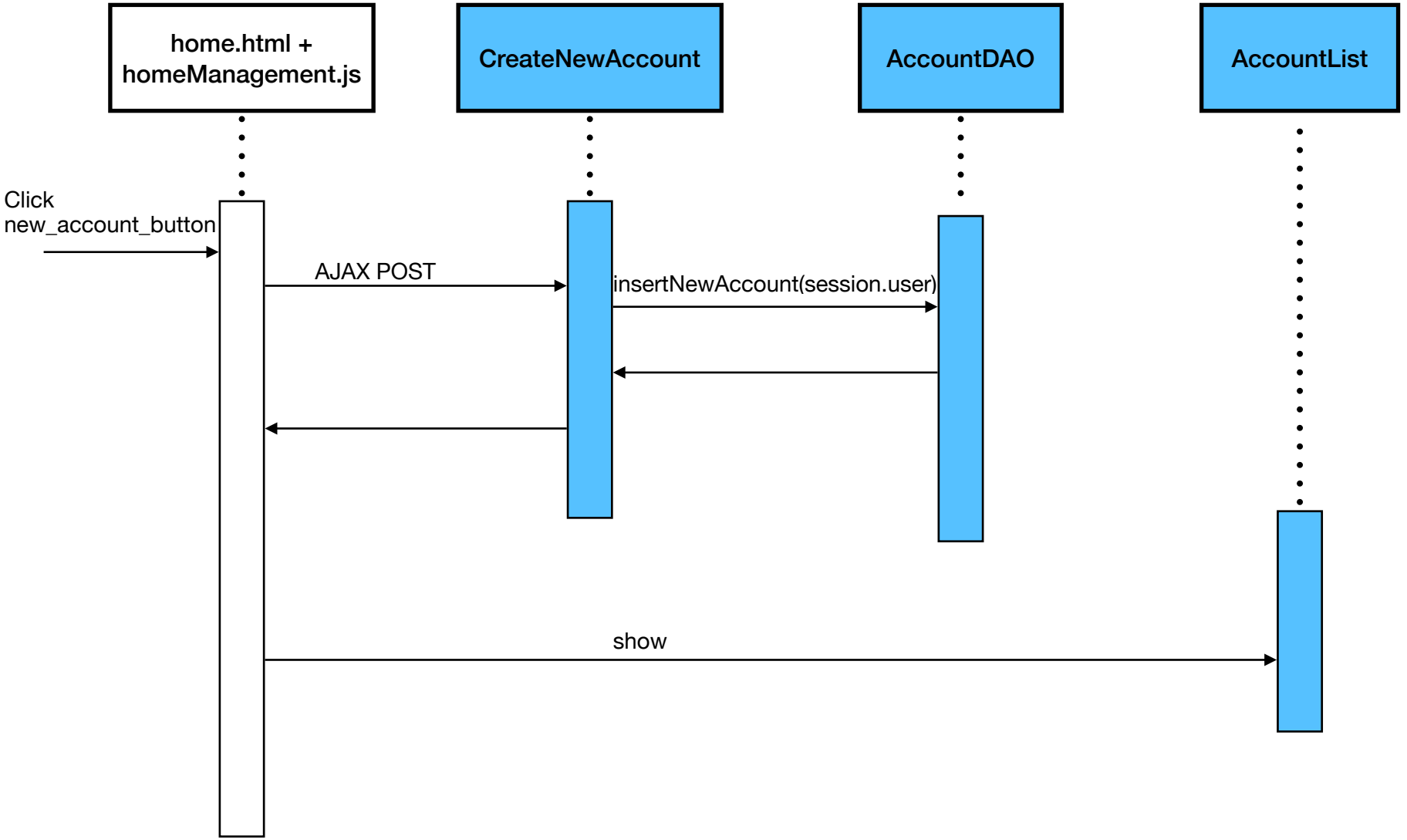
Caricamento HomePage



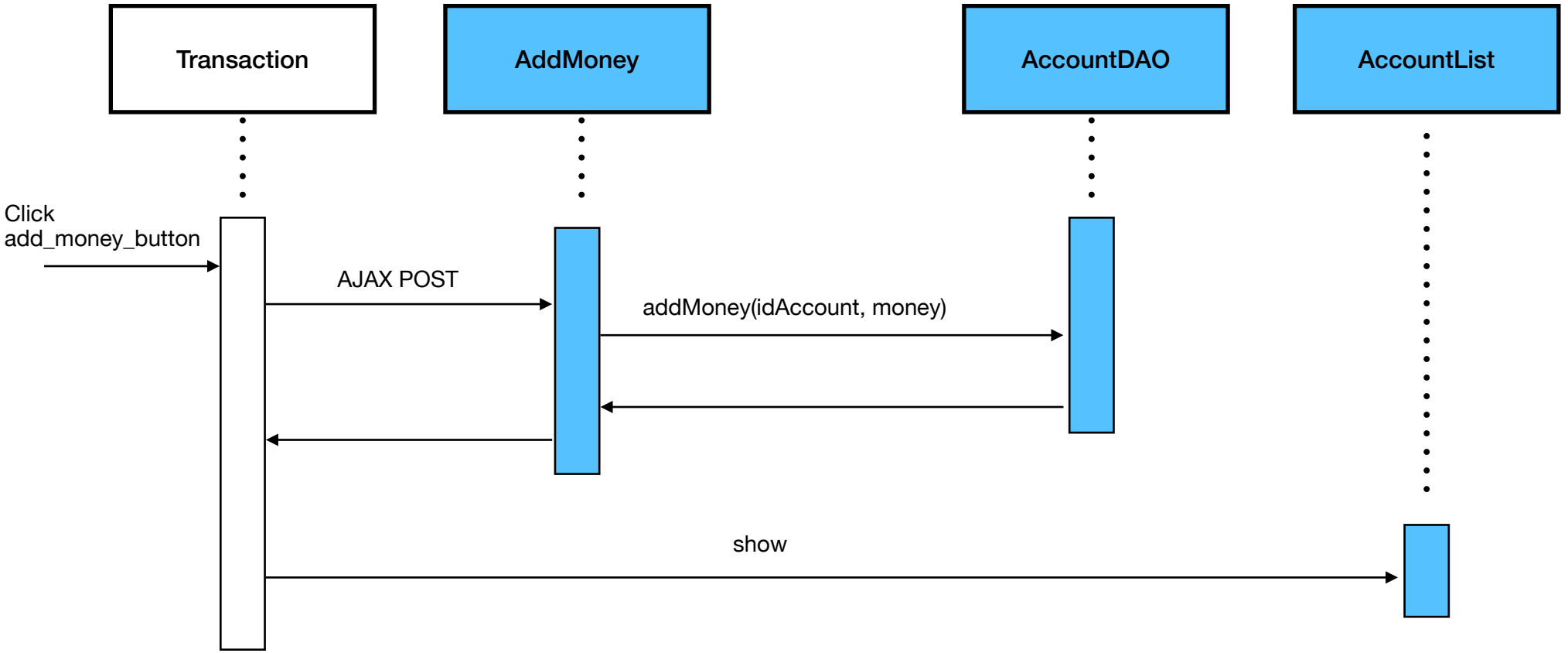
Selezione account



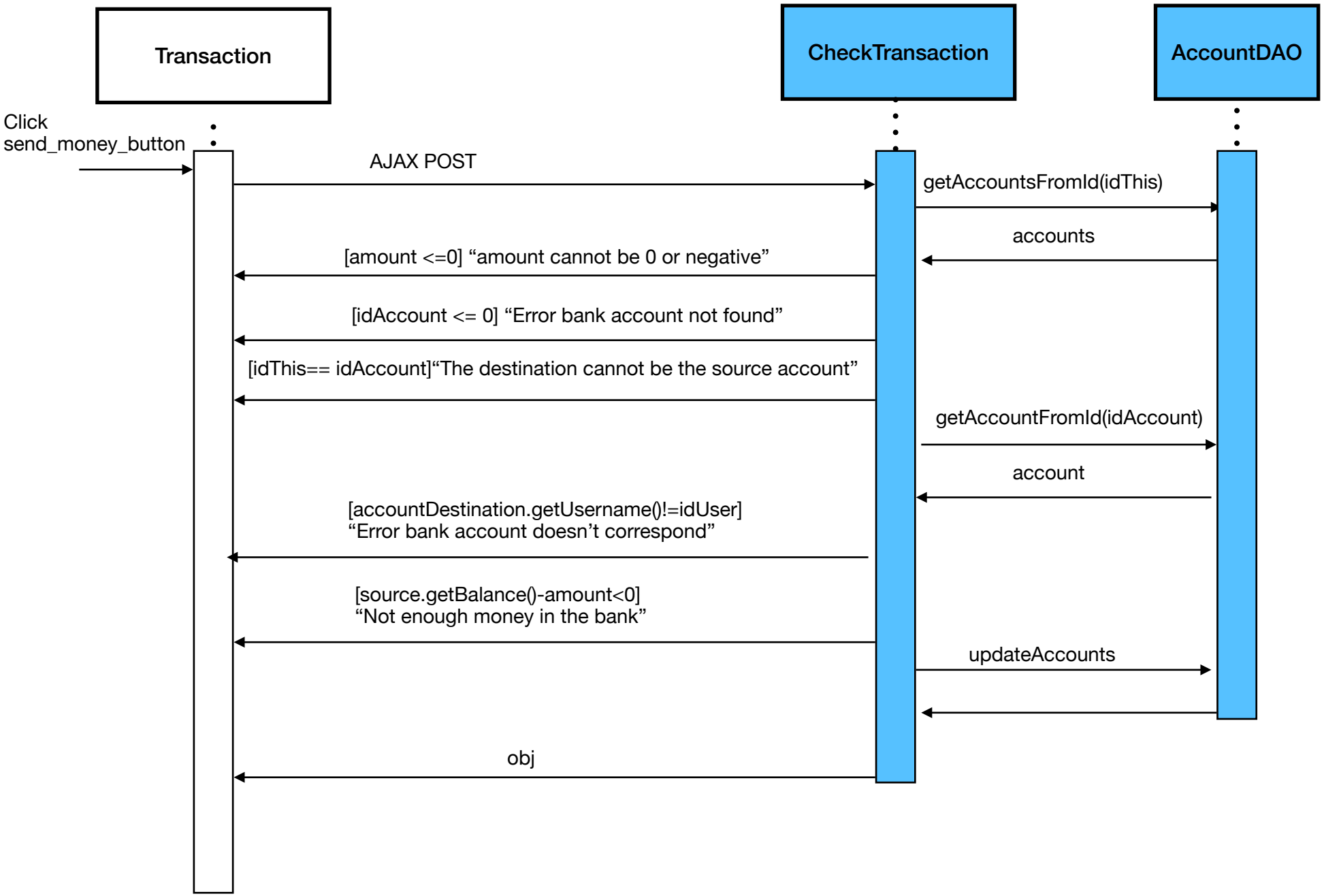
Creazione account

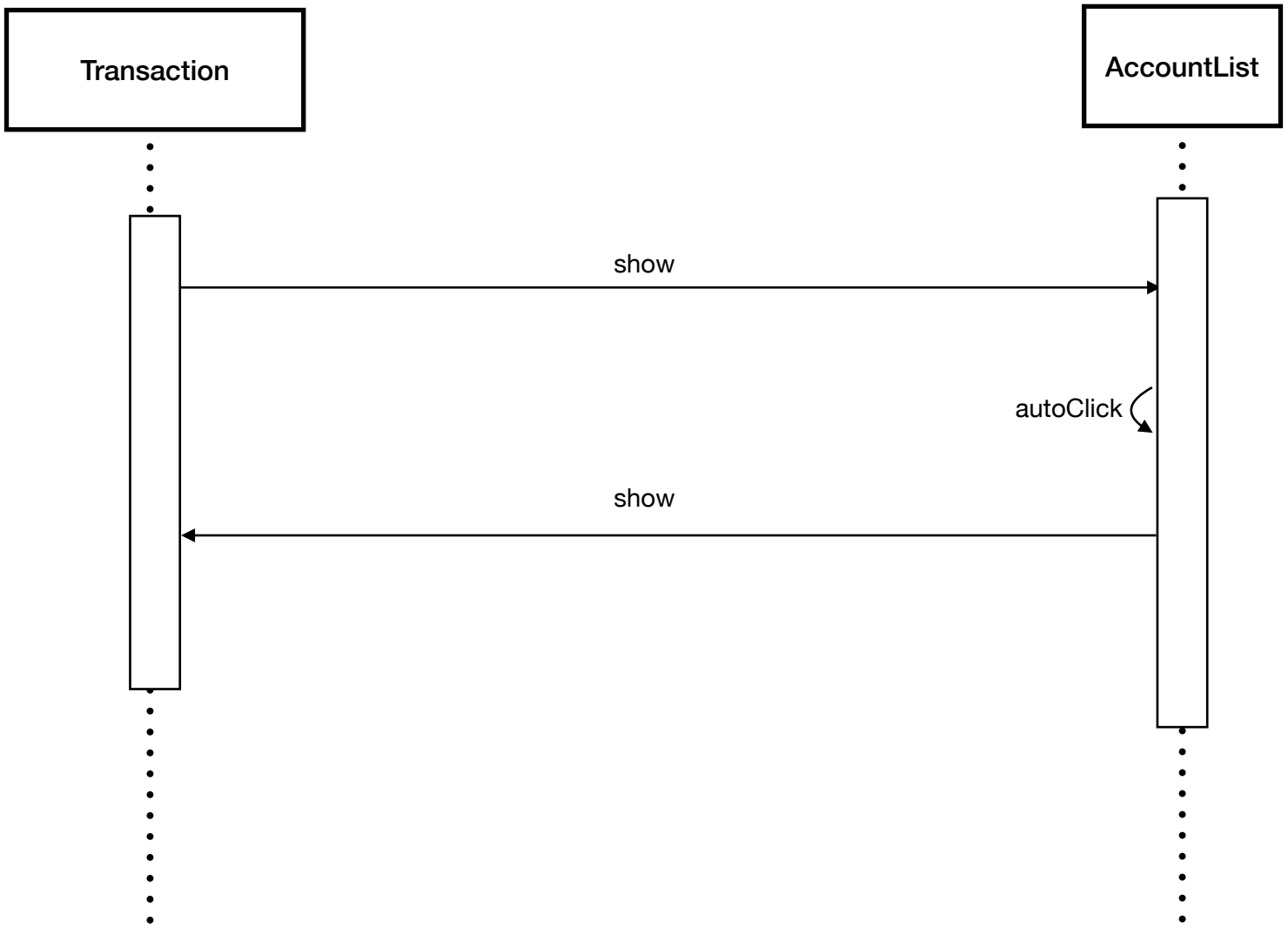


Aggiunta soldi

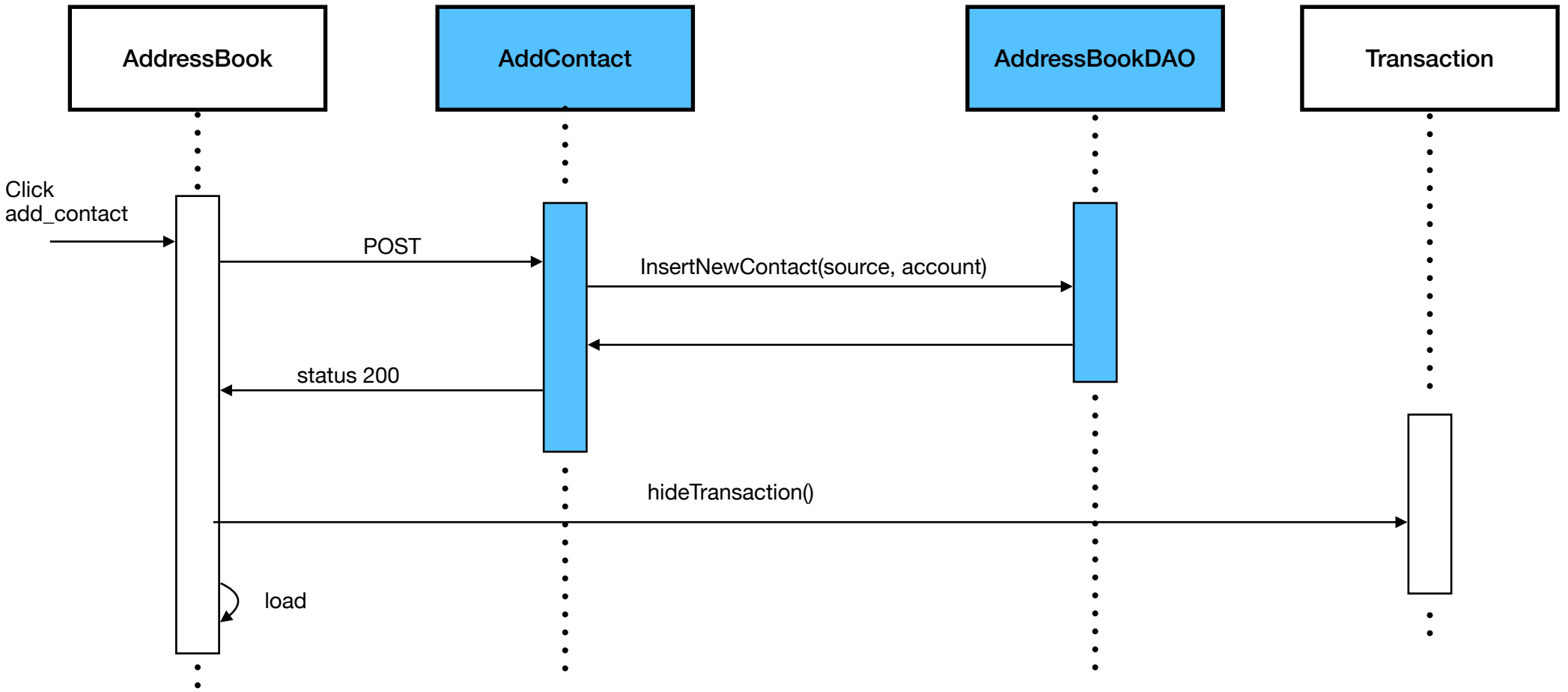


Fai Transazione

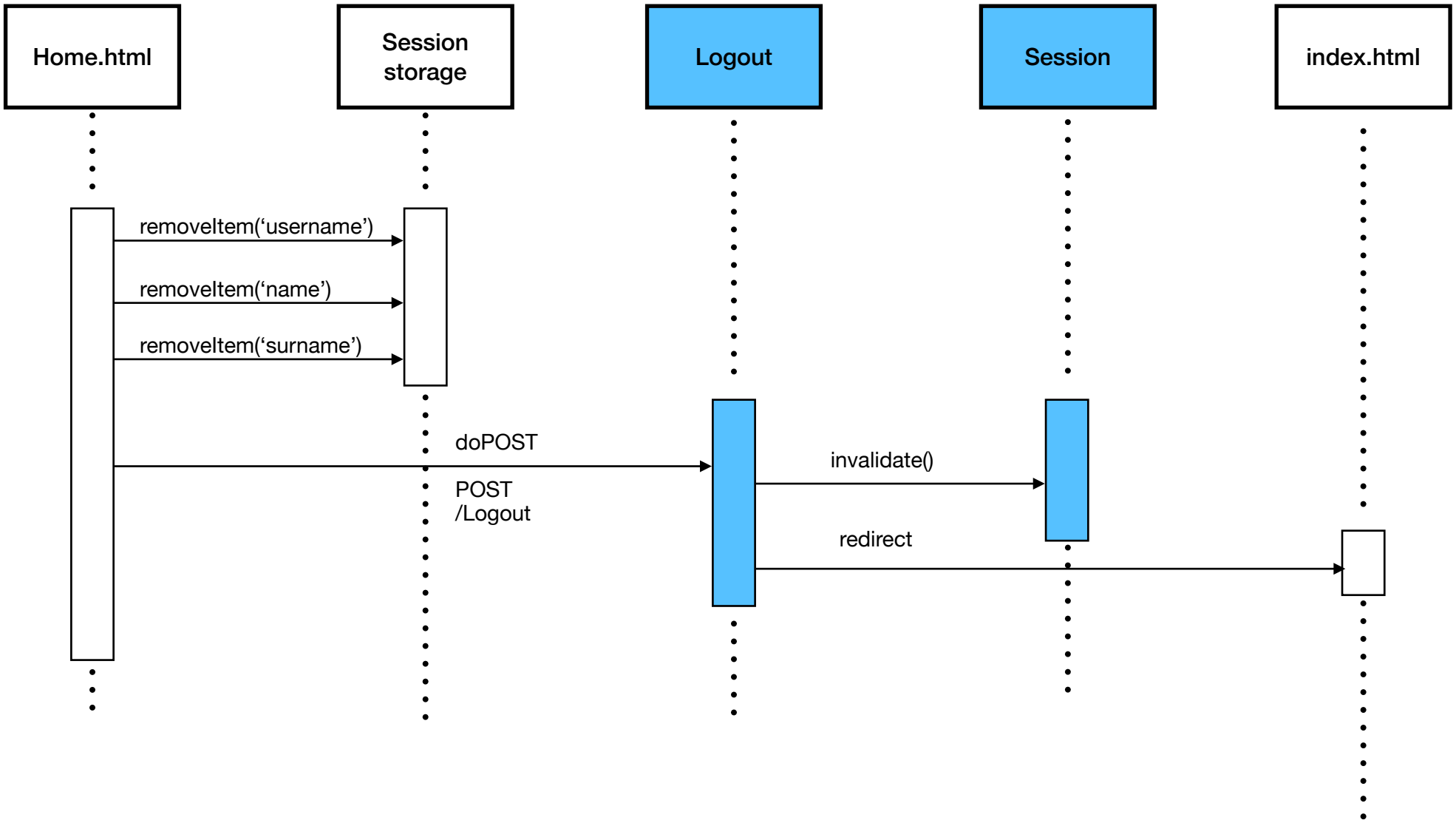



Transaction
AccountList
show
autoClick
show

Aggiungi contatto in rubrica



Logout


Home.html
Session storage
Logout
Session
index.html
removeItem('username')
removeItem('name')
removeItem('surname')
doPOST
POST
/Logout
invalidate()
redirect