Traccia 4 – RIA

Gruppo 66

Federico Toschi – Sara Resta

Indice

Analisi ER

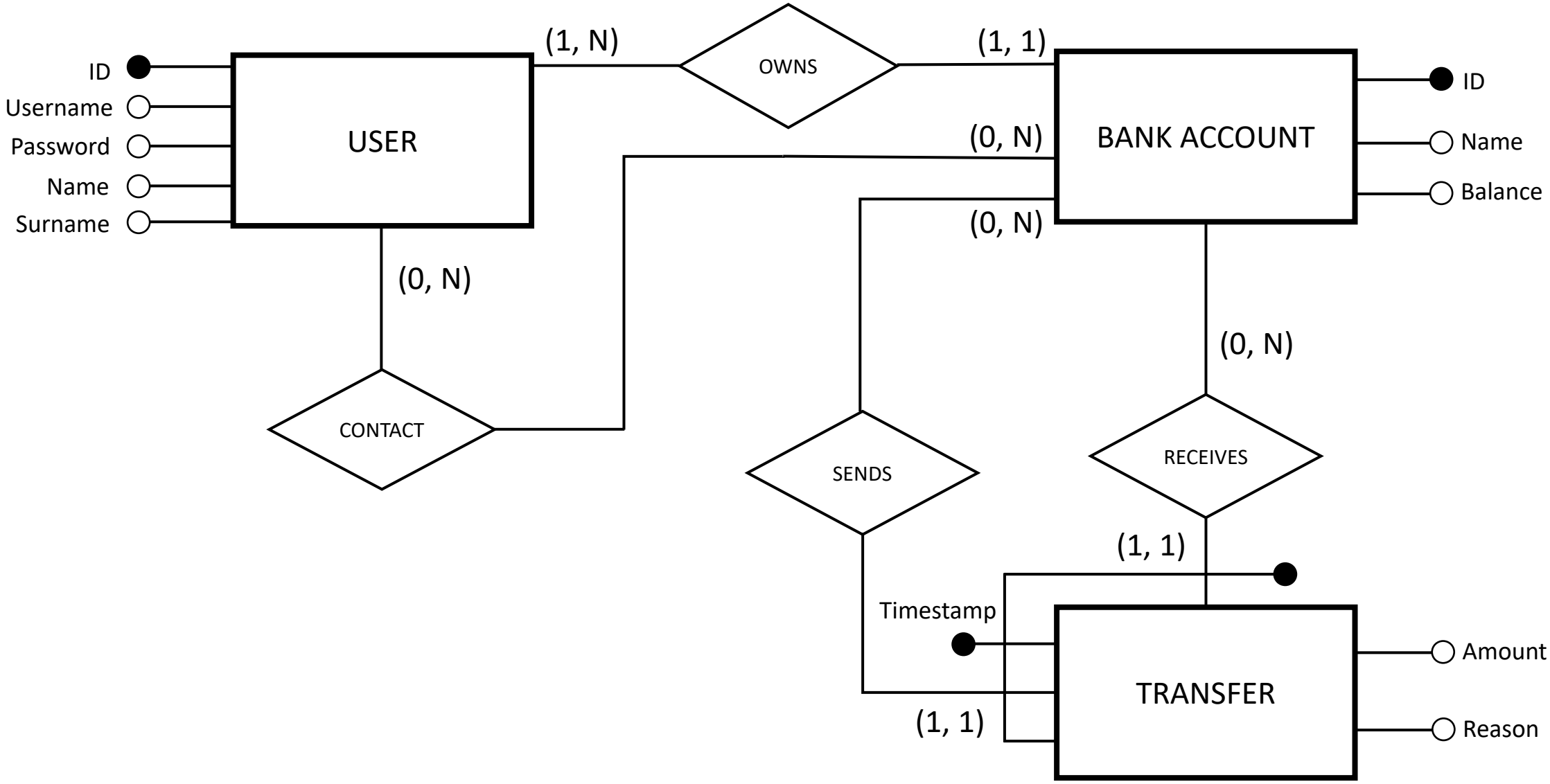
VERSIONE PURE HTML

- 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 desti- nazione 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.
- Entities, Attributes, Relations

VERSIONE RIA

- 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.

Diagramma ER



Database schema (1/4)

```
CREATE TABLE `user` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `name` varchar(20) NOT NULL,
  `surname` varchar(30) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE `username` (`username`)
) ENGINE=InnoDB;
```

Database schema (2/4)

```
CREATE TABLE `bank_account` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `userID` int NOT NULL,
  `name` varchar(40) NOT NULL,
  `balance` decimal(10,2) NOT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `UID` FOREIGN KEY (`userID`)
  REFERENCES `user` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Database schema (3/4)

```
CREATE TABLE `transfer` (
  `senderID` int NOT NULL,
  `recipientID` int NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `reason` varchar(255) NOT NULL,
  `amount` decimal(10,2) NOT NULL,
  PRIMARY KEY (`timestamp`,`senderID`,`recipientID`),
  CONSTRAINT `DAccount` FOREIGN KEY (`recipientID`)
  REFERENCES `bank_account` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `OAccount` FOREIGN KEY (`senderID`)
  REFERENCES `bank_account` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `PositiveAMount` CHECK ((`amount` >= 0))
) ENGINE=InnoDB;
```

Database schema (4/4)

```
CREATE TABLE `contacts` (
  `ownerID` int NOT NULL,
  `accountID` int NOT NULL,
  PRIMARY KEY (`ownerID`,`accountID`),
  CONSTRAINT `accountID` FOREIGN KEY (`accountID`)
  REFERENCES `bank_account` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `ownerID` FOREIGN KEY (`ownerID`)
  REFERENCES `user` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Analisi dei requisiti

- 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.
- Pages (Views), View Components, Events, Actions

VERSIONE RIA

- 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.

Completamento delle specifiche

- L’utente è identificato tramite ID o username (che coincide con l’indirizzo e-mail).
- Per credenziali di login si intendono email e password.
- Ogni utente deve possedere almeno un conto corrente, al momento della registrazione viene creato un conto corrente di default avente saldo pari a zero.
- Dalla pagina Home è possibile creare un nuovo conto specificandone il nome. Il nuovo conto creato avrà saldo pari a zero. Non sono ammessi nomi duplicati.
- Se l’utente ha effettuato il login e tenta di accedere alla pagina di Login/Register verrà reindirizzato automaticamente alla HOME. Viceversa se non ha effettuato il login e tenta di accedere a pagine protette verrà reindirizzato alla pagina di LOGIN
- Se si verificano azioni impreviste, viene mostrata la descrizione dell’errore a schermo tramite alert/error div.
- Non è possibile richiedere trasferimenti con conto origine uguale al conto destinazione.
- Ogni stringa inserita dall’utente viene sanificata lato server tramite escaping.
- Viene effettuato unescaping di ogni stringa prima dell’invio all’utente.
- L’utente non può aggiungere ai propri contatti uno dei suoi conti.
- L’autocompletamento mostra solamente i conti aggiunti ai contatti dall’utente (ed i relativi user ID). All’inserimento di uno userID, l’utente vedrà suggerimenti solo relativi ad eventuali conti dello userID precedentemente inseriti tra i contatti.

Server-side Components

- Model objects: beans
  - User
  - BankAccount
  - Transfer
  - Contacts
- Data Access Object (DAO)
  - UserDAO
    - findUser(String email, String password): User
    - findUserByID(int ID): User
    - registerUser (String email, String password, String name, String surname): int
    - isEmailTaken(String email): boolean
  - BankAccountDAO
    - findAccountByID(int ID): BankAccount
    - findAllAccountsByUserID (int userID): List<BankAccount>
    - createAccount(int userid, String name, BigDecimal balance)
    - isNameTaken(int userID, String name): boolean
  - TransferDAO
    - getTransferByAccountID (int accountID): List<Transfer>
    - makeTransfer(BigDecimal amount, String reason, int senderid, int recipentid)
    - getLastTransferByUserID(int userID): Transfer
  - ContactsDAO
    - getContactsByUserID(int accountID)
    - insertContact(int accountID, int contactID)
- Filters
  - LoginFilter
  - NotLoggedFilter
- Controllers (servlets) [access rights]
  - Login [not logged user]
  - Register [not logged user]
  - Logout [*]
  - SelectAccount [logged user]
  - MakeTransfer [logged user]
  - GetContacts [logged user]
  - GetAccounts [logged user]
  - InsertContact [logged user]
- Views
  - Login
  - Home

Client-side Components

- **Login**
  - Login form
  - Register form
  - Wizard
- **Home**
  - PageOrchestrator
  - User data
  - CreateAccount form
  - Account list
  - Transfer list
  - MakeTransfer form
  - TransferResult
  - Contacts

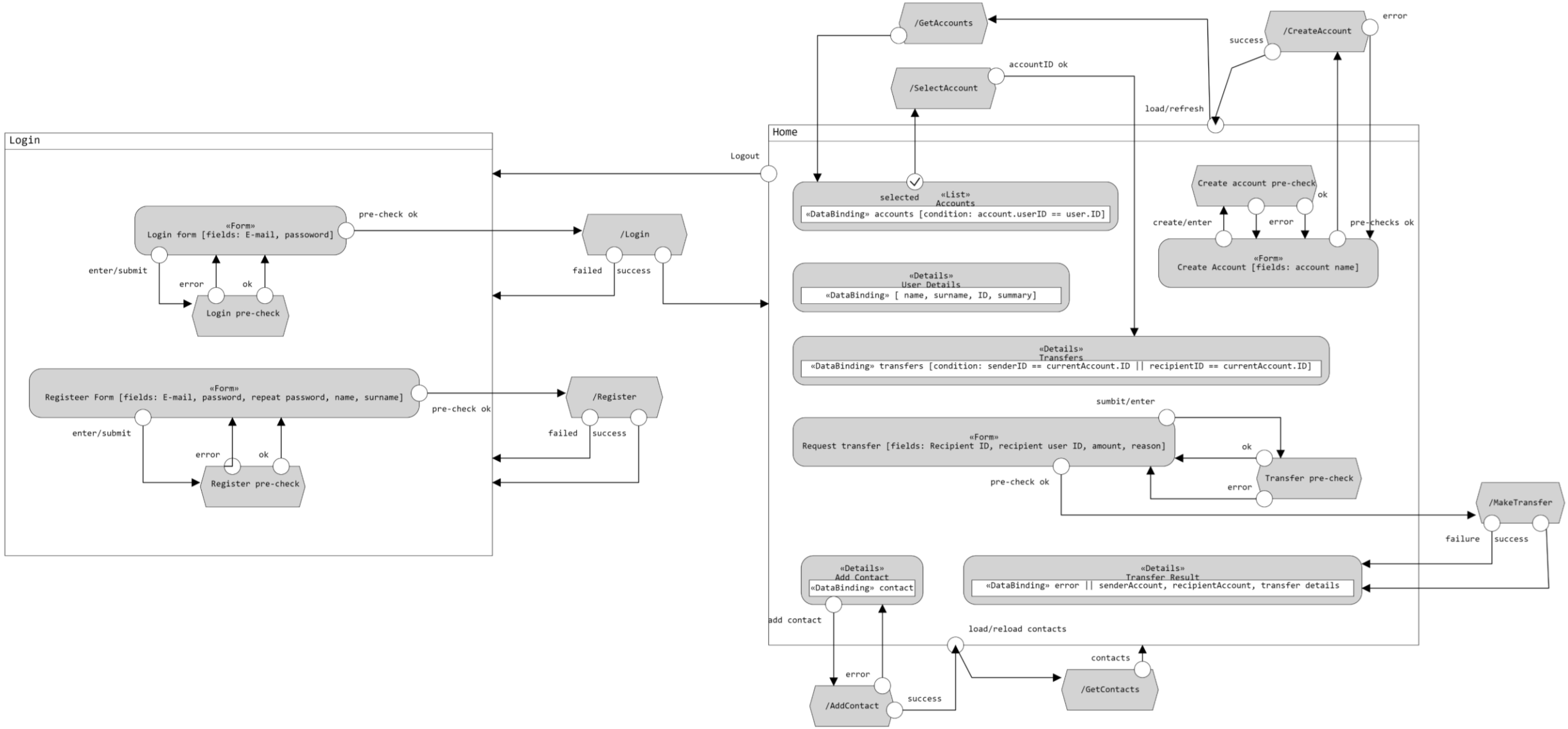
Events & Actions

Client side		Server side	
Event	**Action**	**Event**	**Action**
Login.html → login form → submit	Credential check (e-mail syntax check, password min 8 characters)	POST (username, password)	Credentials check
Login.html → register form → submit	Data check (e-mail syntax check, password min 8 characters, matching passwords, checks on name & surname encoding)	POST (username, password, repeat password, name, surname)	Data check
Login.html → Register button	Change form	/	/
Login.html → Login button	Change form	/	/
Home.html → load	Update view, show transfers of default account, make transfers	GET	Retrive data from data base + json conversion
Home.html → Create account form → Submit	Input check	POST	Input check
Home.html → AccountList → Account selection	Update view, show transfers, show make transfer button	GET(accountID)	Retrive data from data base + json conversion
Home.html → TransferForm → Submit transfer	Input check	POST(Transfer data)	Transfer checks, Return outcome
Home.html → TransferForm → insert input	Autocomplete	/	/
Home.html → TransferResult → Success	Optional add contact button	/	/
Home.html → TransferResult → Success→ Add contact	Add contact (AJAX POST)	POST(ownerID, contactID)	Add contact
Logout		GET	Invalidate session

Controller / event handler

Client side		Server side	
Event	**Controller**	**Event**	**Controller**
Login.html → login form → submit	Function makeCall	POST (email, password)	Login (servlet)
Login.html → register form → submit	Function makeCall	POST (email, password, repPassword, name, surname)	Register (servlet)
Login.html → login form → Register button	Wizard.show(false)	/	/
Login.html → register form → Login	Wizard.show(true)	/	/
Home.html → load	Function PageOrchestrator…	GET without params GET without params	GetAccounts (servlet) GetContacts (servlet)
Home.html → Create account form → Submit	Fuction makeCall	POST (accountName)	CreateAccount (servlet)
Home.html → AccountList → Account selection	Function AccountDetails.show(accountID)	GET (accountID)	SelectAccount (servlet)
Home.html → TransferForm → Submit transfer	Function makeCall	POST (senderID, recipientUserID, recipientID, amount, reason)	MakeTransfer(servlet)
Home.html → TransferForm→ insert input	Contacts. autocompleteRecipientUserID (recipientUserID) Contacts. autocompleteRecipientAccountID (recipientUserID, recipientAccountID, currentAccountID)	/	/
Home.html → TransferResult → Add contact	Function makeCall	POST (contactID)	AddContact (servlet)
Logout		GET	Logout (servlet)

Design Applicativo - IFML



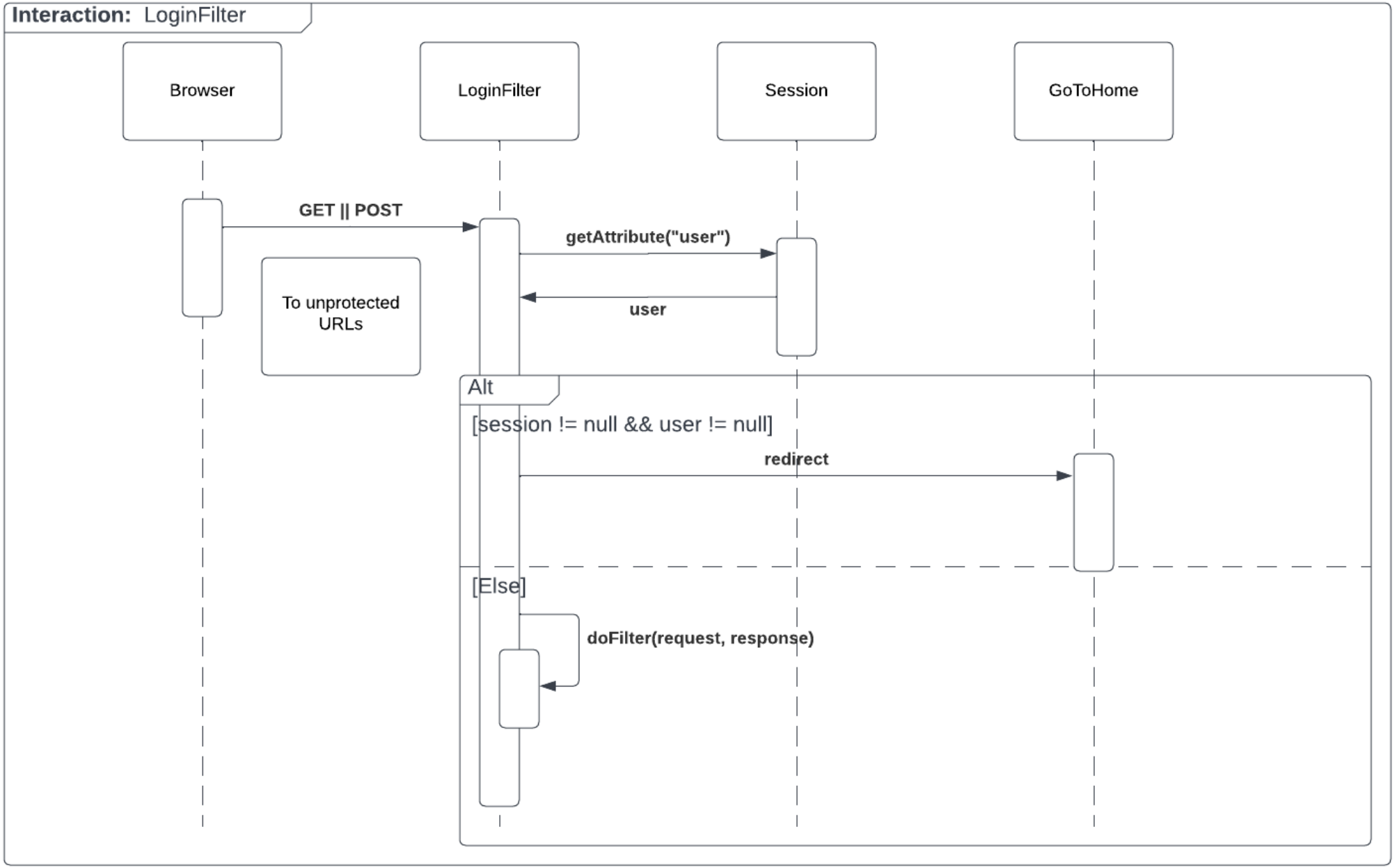
Sequence diagrams

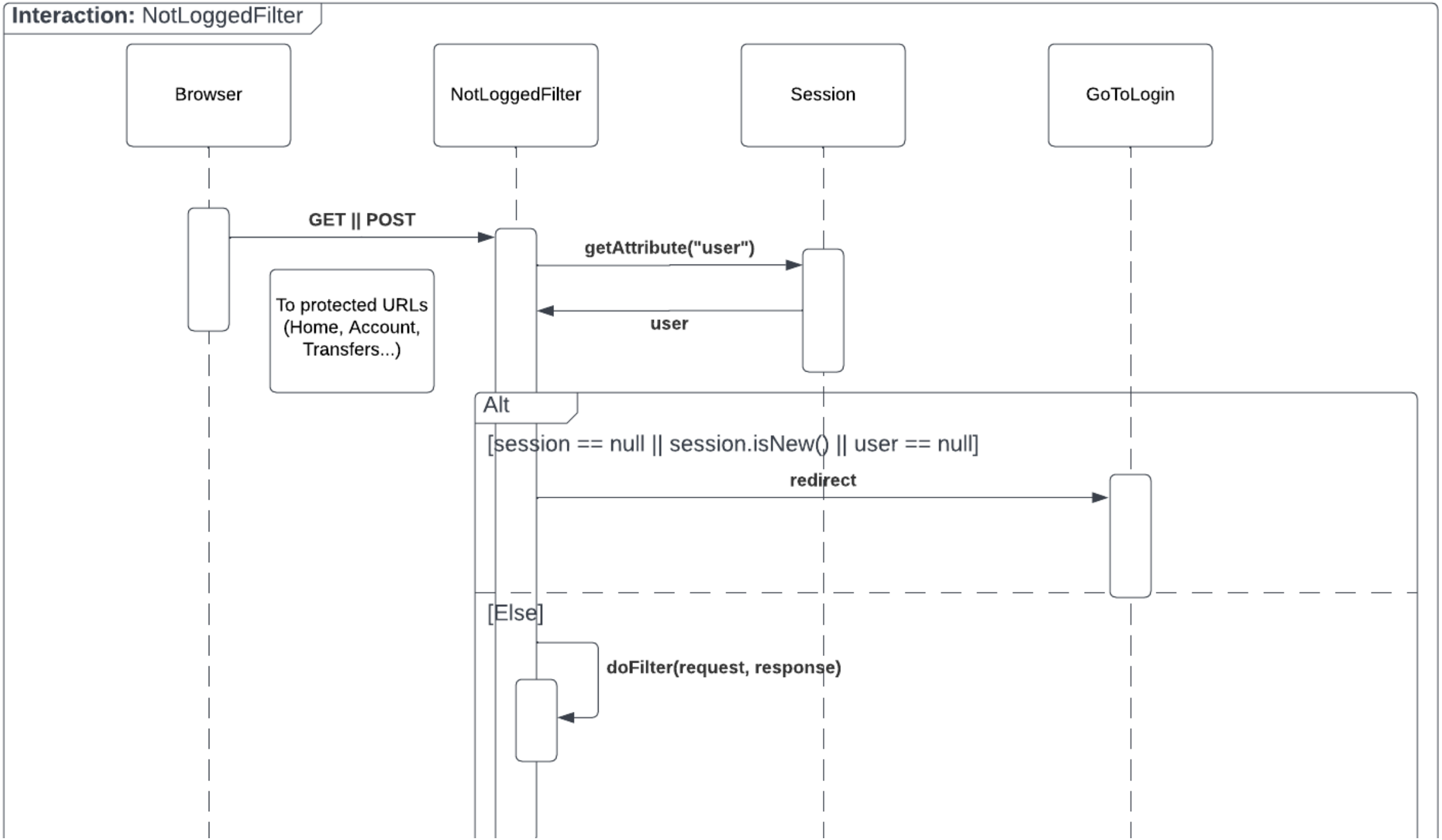
The following sequence diagrams aim to depict the interaction flow underlying the core events of the web application. Although the authors tried their best to pursue and achieve full clarity, some minor details have been left out due to their repetitiveness (e.g. Internal server errors, Database access errors, etc.).

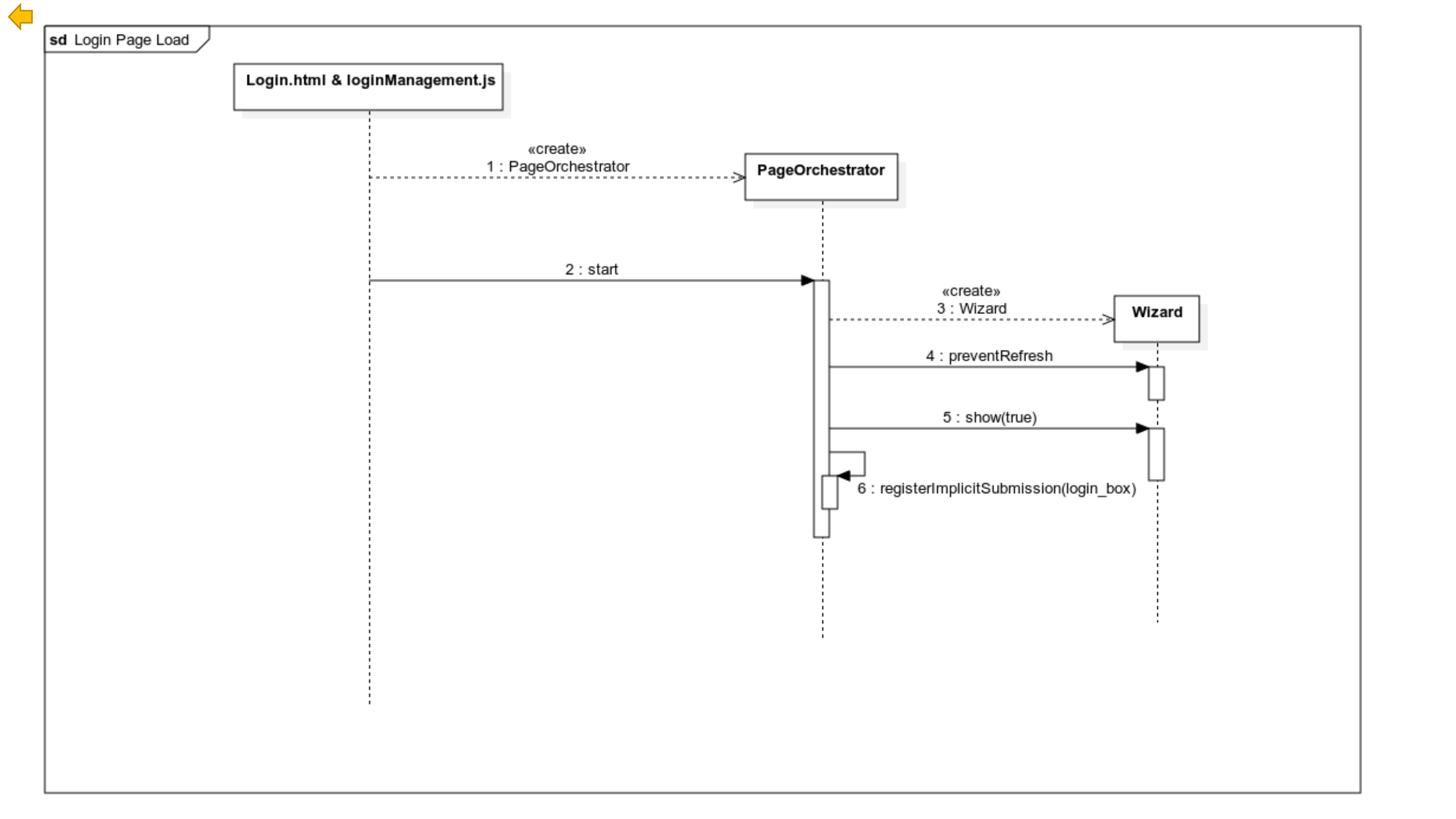
Filter checks will be represented in the first diagrams and omitted in the subsequent ones.

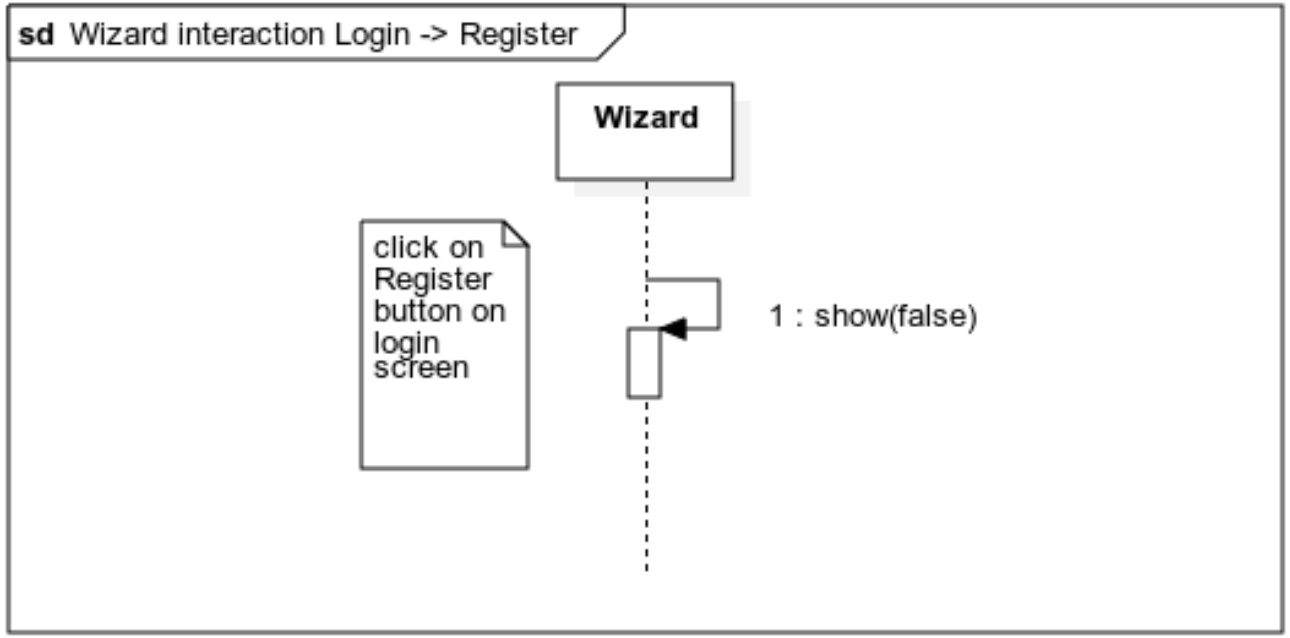
Page loading sequences will include a simplified representation of AJAX calls. Those interactions will get a more detailed representation in their own diagrams, which can be viewed by clicking on the server-side entity.

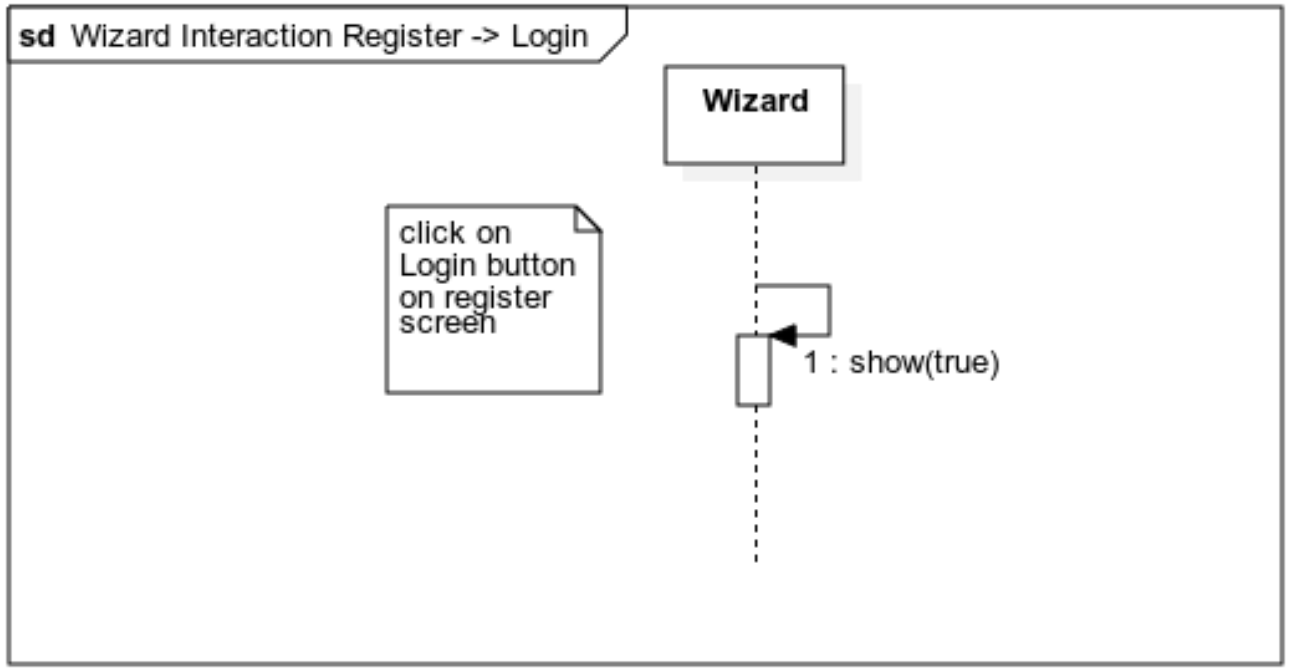
Blue lifelines indicate server-side components, white lifelines indicate client-side components.

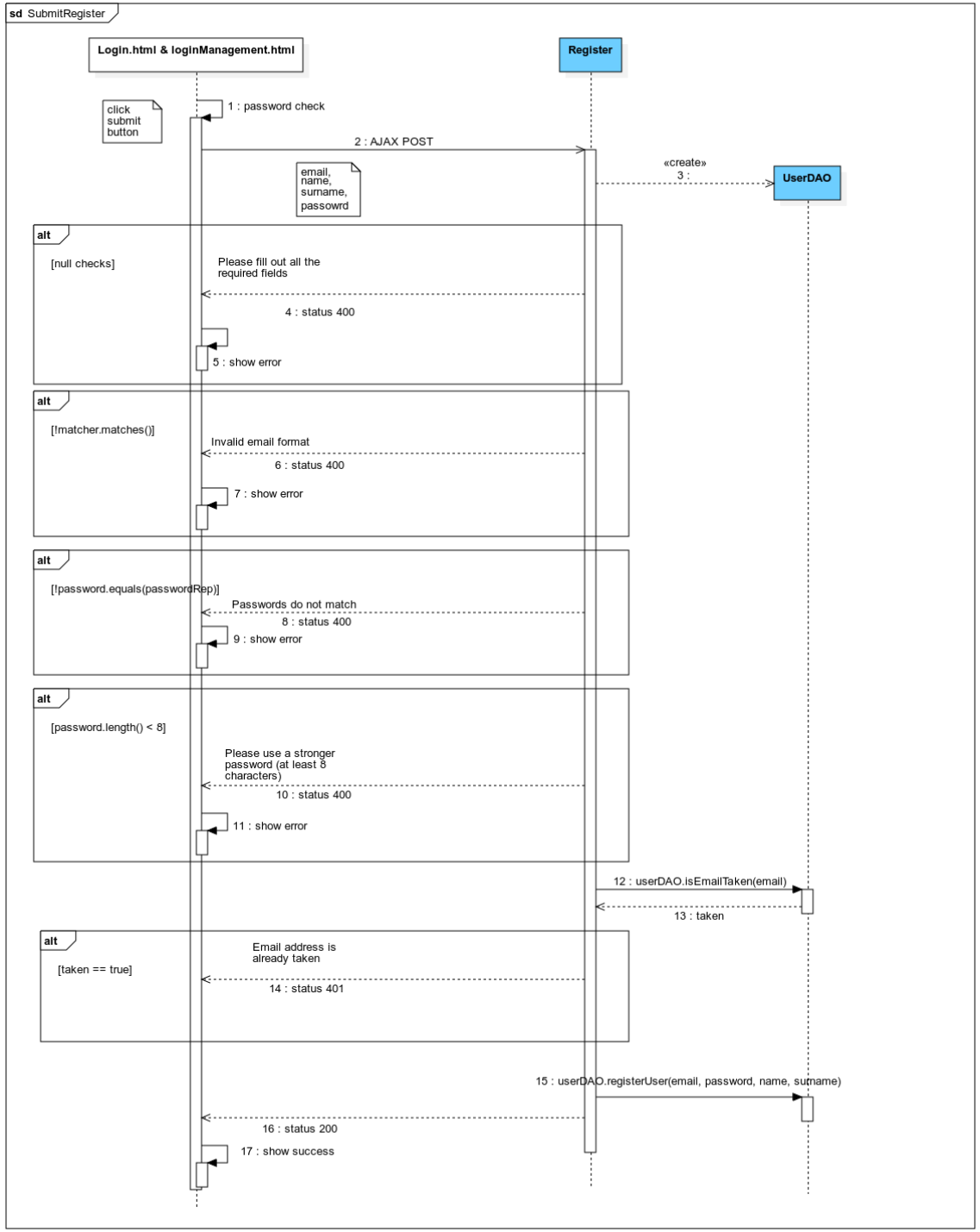

Interaction: LoginFilter
Browser
LoginFilter
Session
GoToHome
GET || POST
getAttribute("user")
To unprotected URLs
user
Alt
[session != null && user != null]
redirect
[Else]
doFilter(request, response)

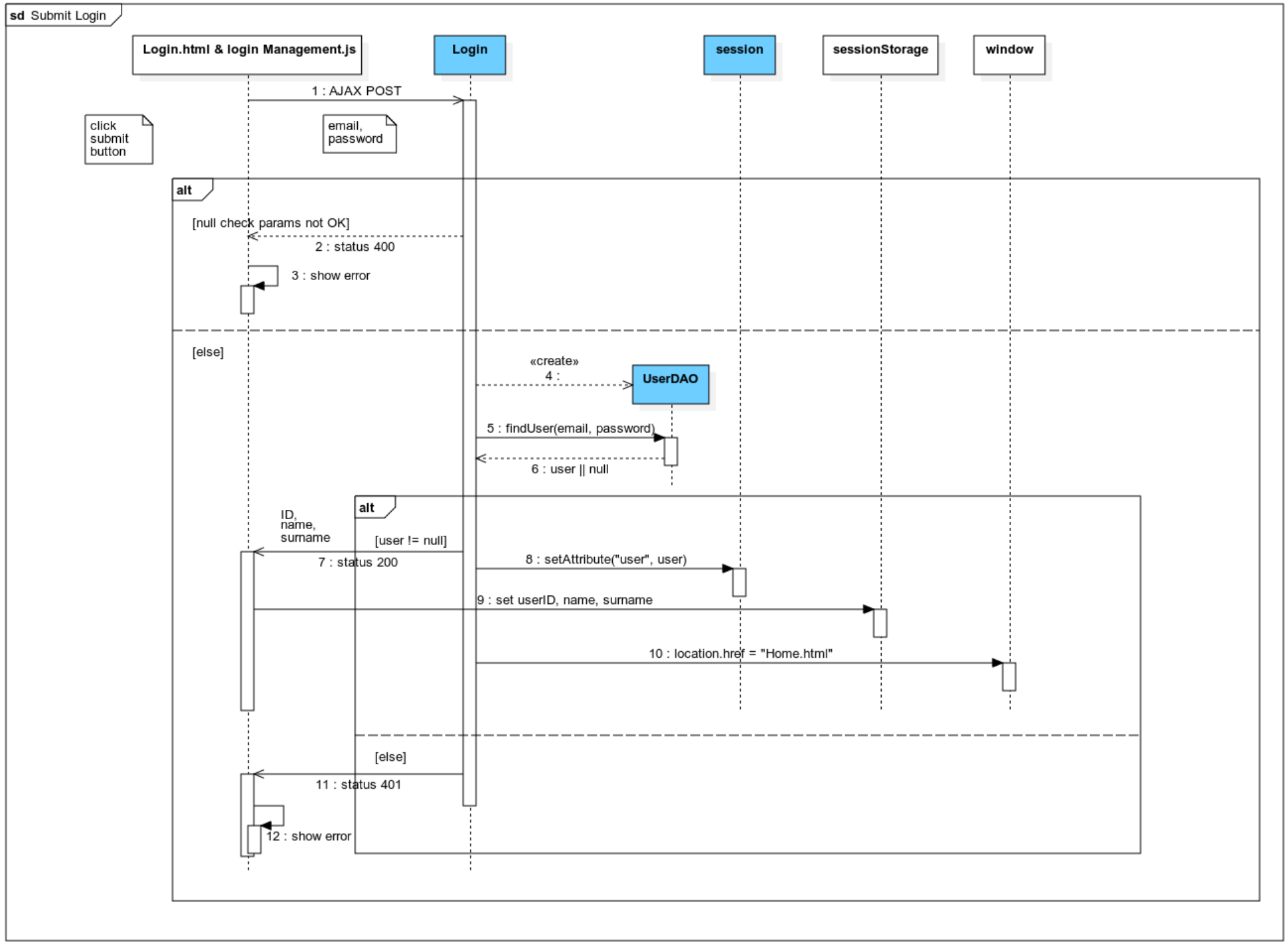

Interaction: NotLoggedFilter
Browser
NotLoggedFilter
Session
GoToLogin
GET || POST
getAttribute("user")
To protected URLs (Home, Account, Transfers...)
user
Alt
[session == null || session.isNew() || user == null]
redirect
[Else]
doFilter(request, response)

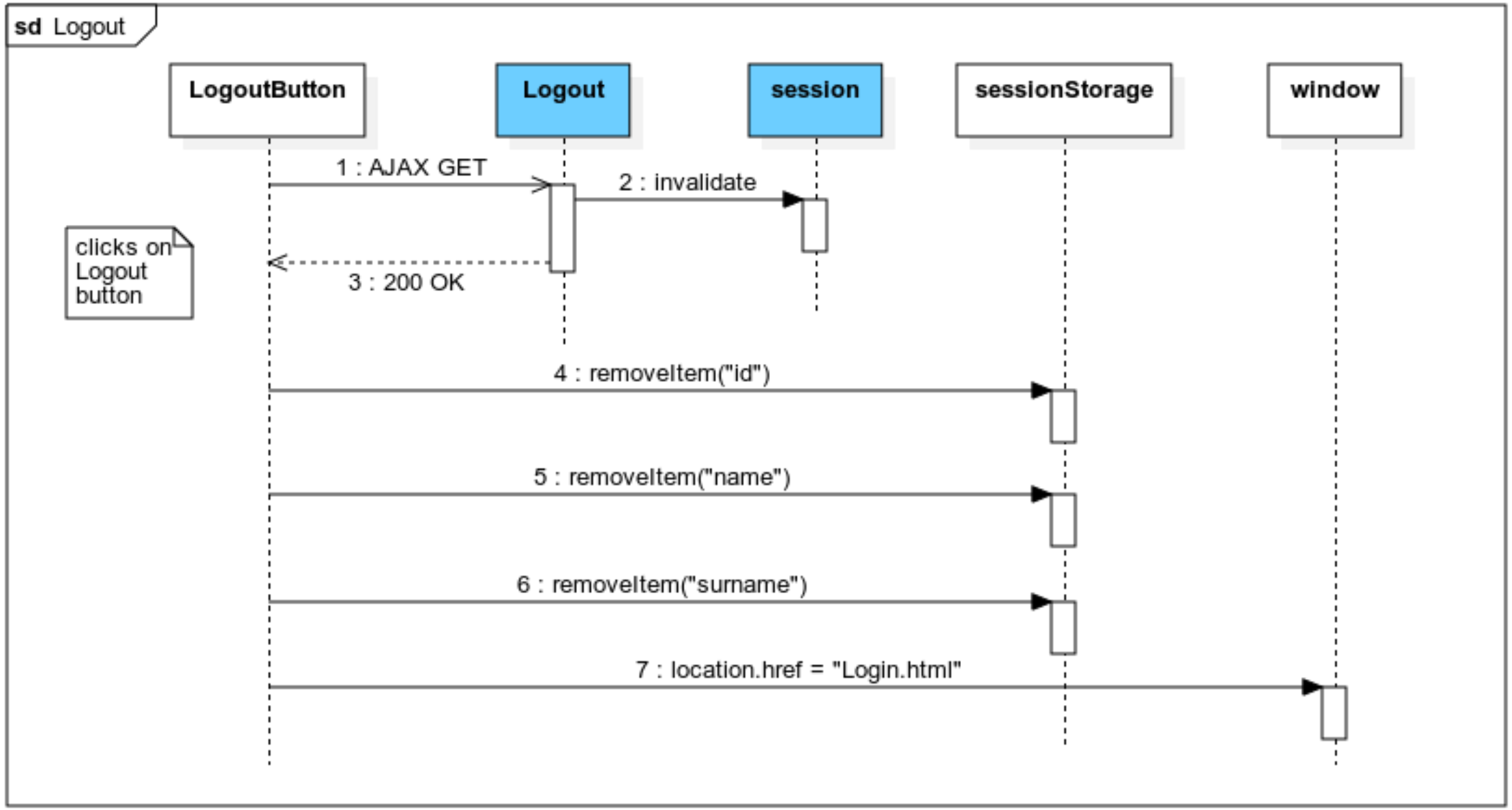

sd Login Page Load
Login.html & loginManagement.js
«create»
1 : PageOrchestrator
PageOrchestrator
2 : start
«create»
3 : Wizard
Wizard
4 : preventRefresh
5 : show(true)
6 : registerImplicitSubmission(login_box)

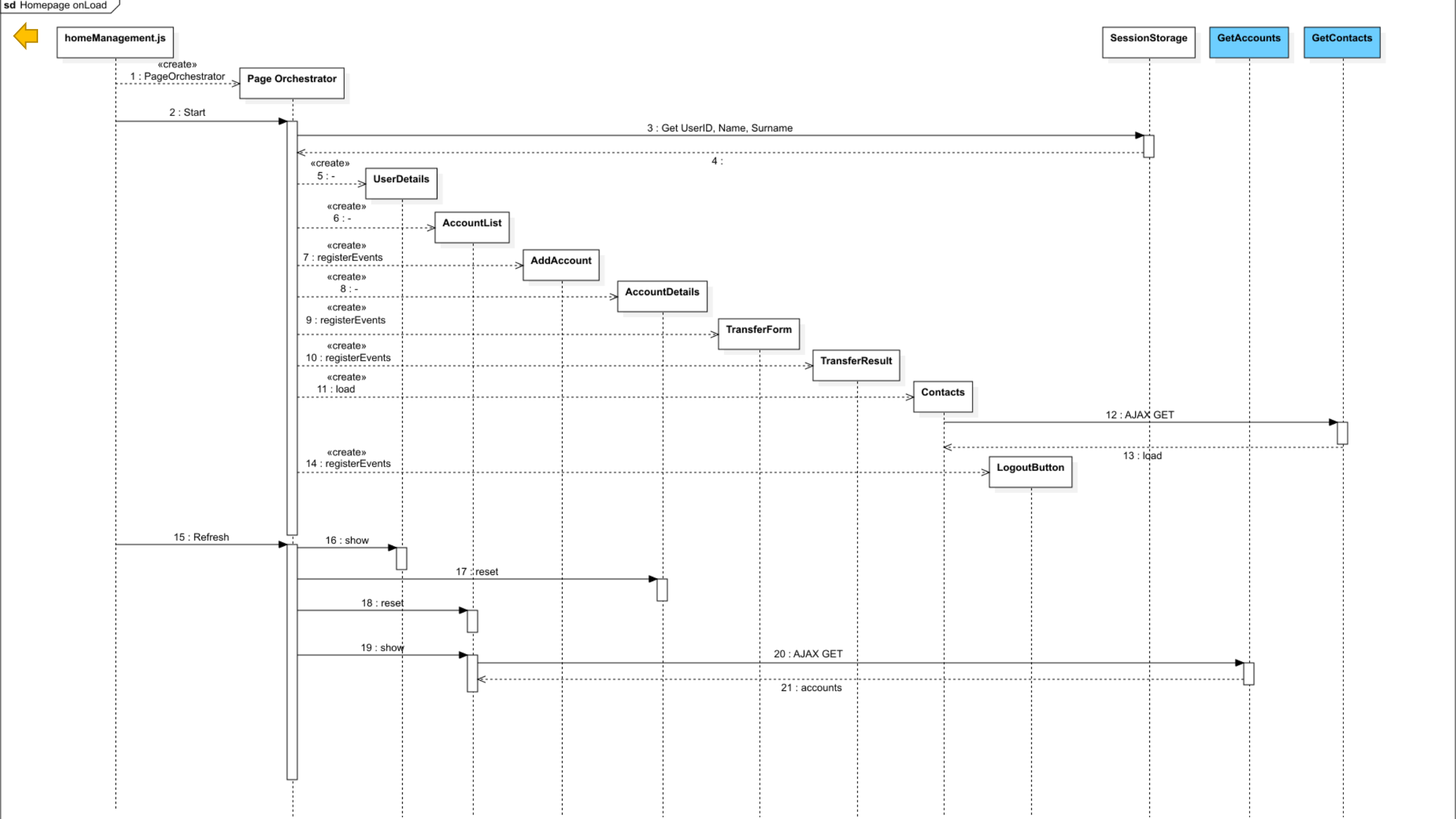

sd Wizard interaction Login -> Register
Wizard
click on Register button on login screen
1 : show(false)

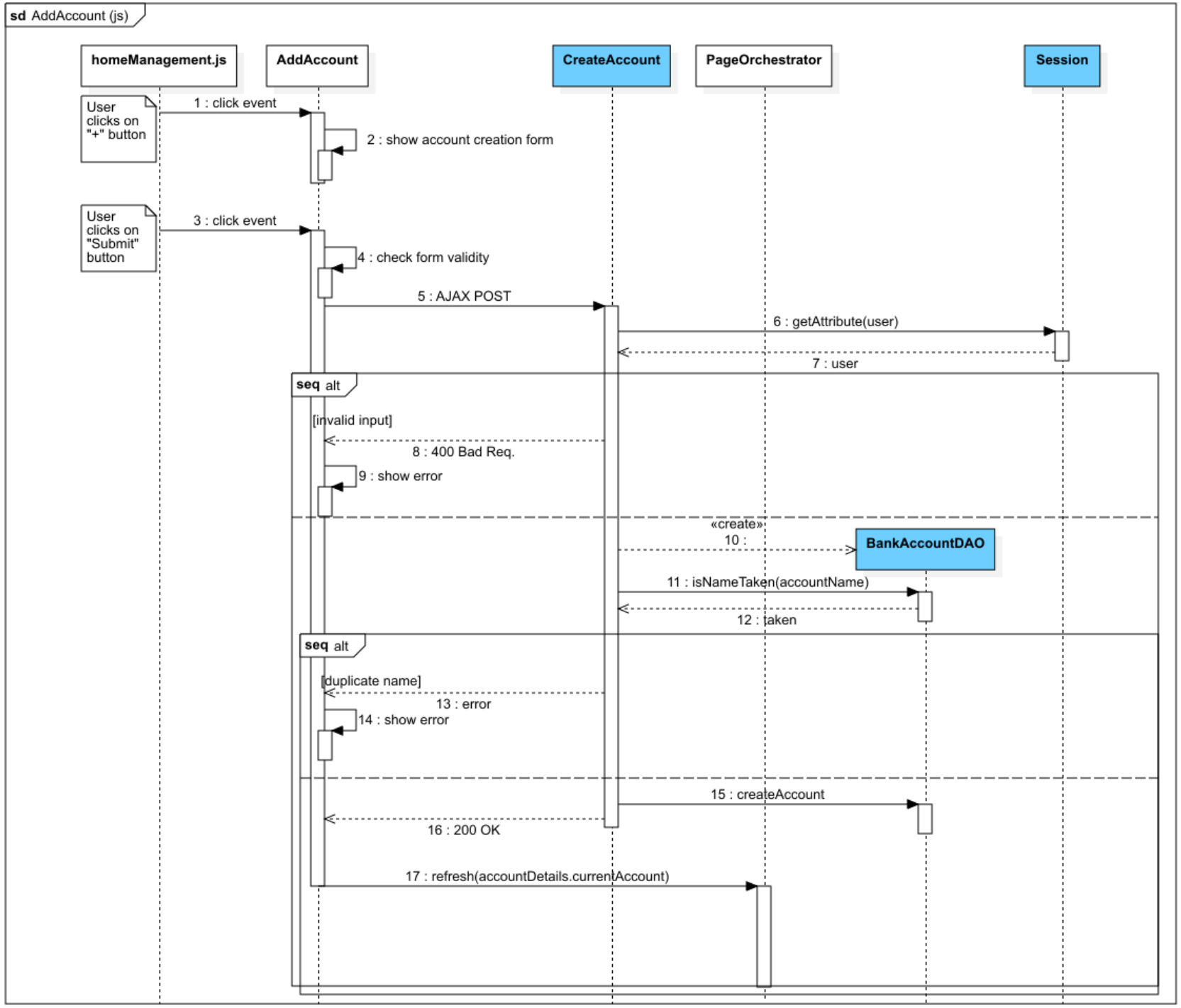

sd Wizard Interaction Register -> Login
Wizard
click on Login button on register screen
1 : show(true)

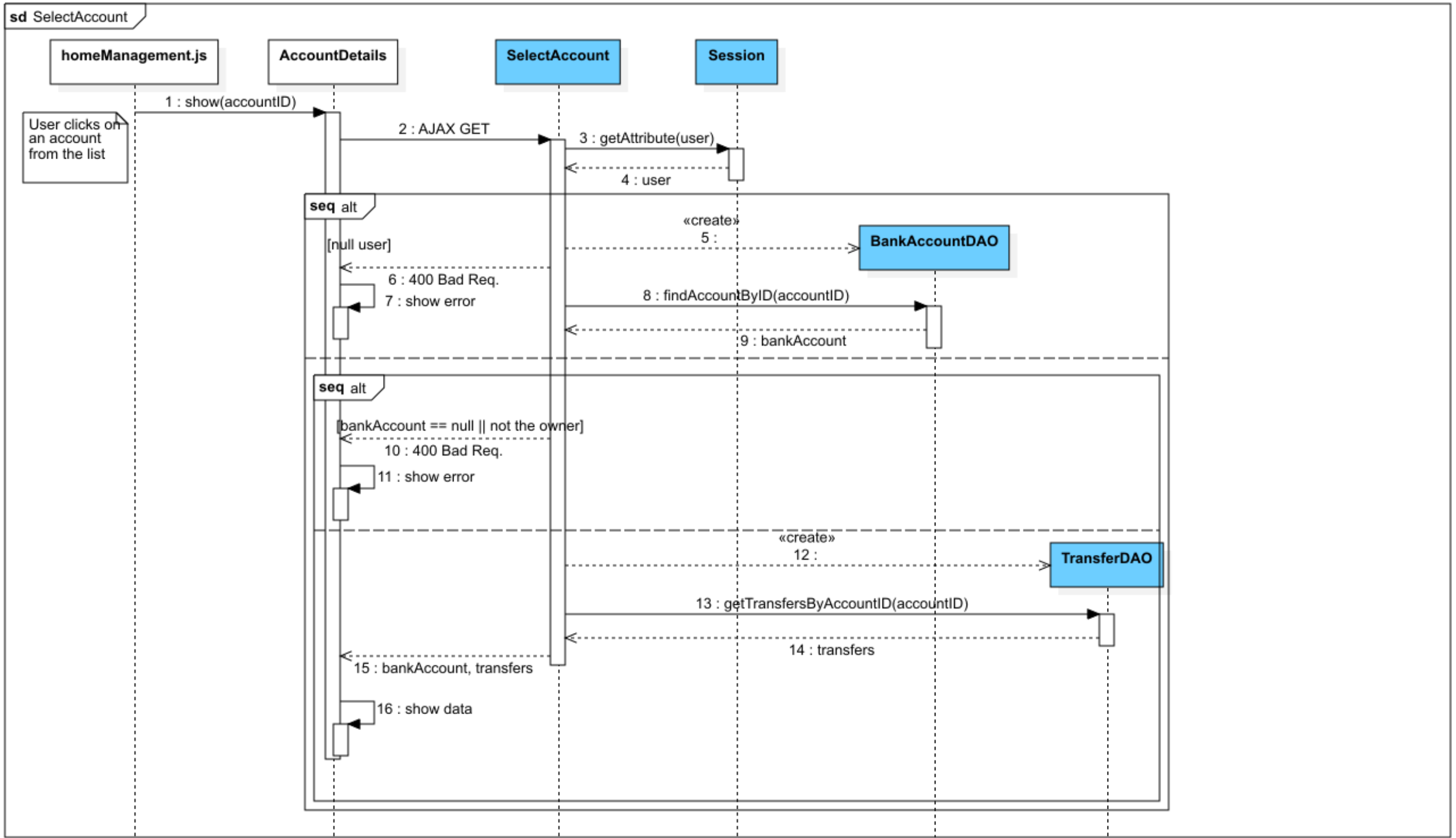

sd SubmitRegister
Login.html & loginManagement.html
Register
UserDAO
click submit button
1 : password check
2 : AJAX POST
email, name, surname, passowrd
«create»
3 :
alt
[null checks]
Please fill out all the required fields
4 : status 400
5 : show error
alt
[!matcher.matches()]
Invalid email format
6 : status 400
7 : show error
alt
[!password.equals(passwordRep)]
Passwords do not match
8 : status 400
9 : show error
alt
[password.length() < 8]
Please use a stronger password (at least 8 characters)
10 : status 400
11 : show error
12 : userDAO.isEmailTaken(email)
13 : taken
alt
[taken == true]
Email address is already taken
14 : status 401
15 : userDAO.registerUser(email, password, name, surname)
16 : status 200
17 : show success

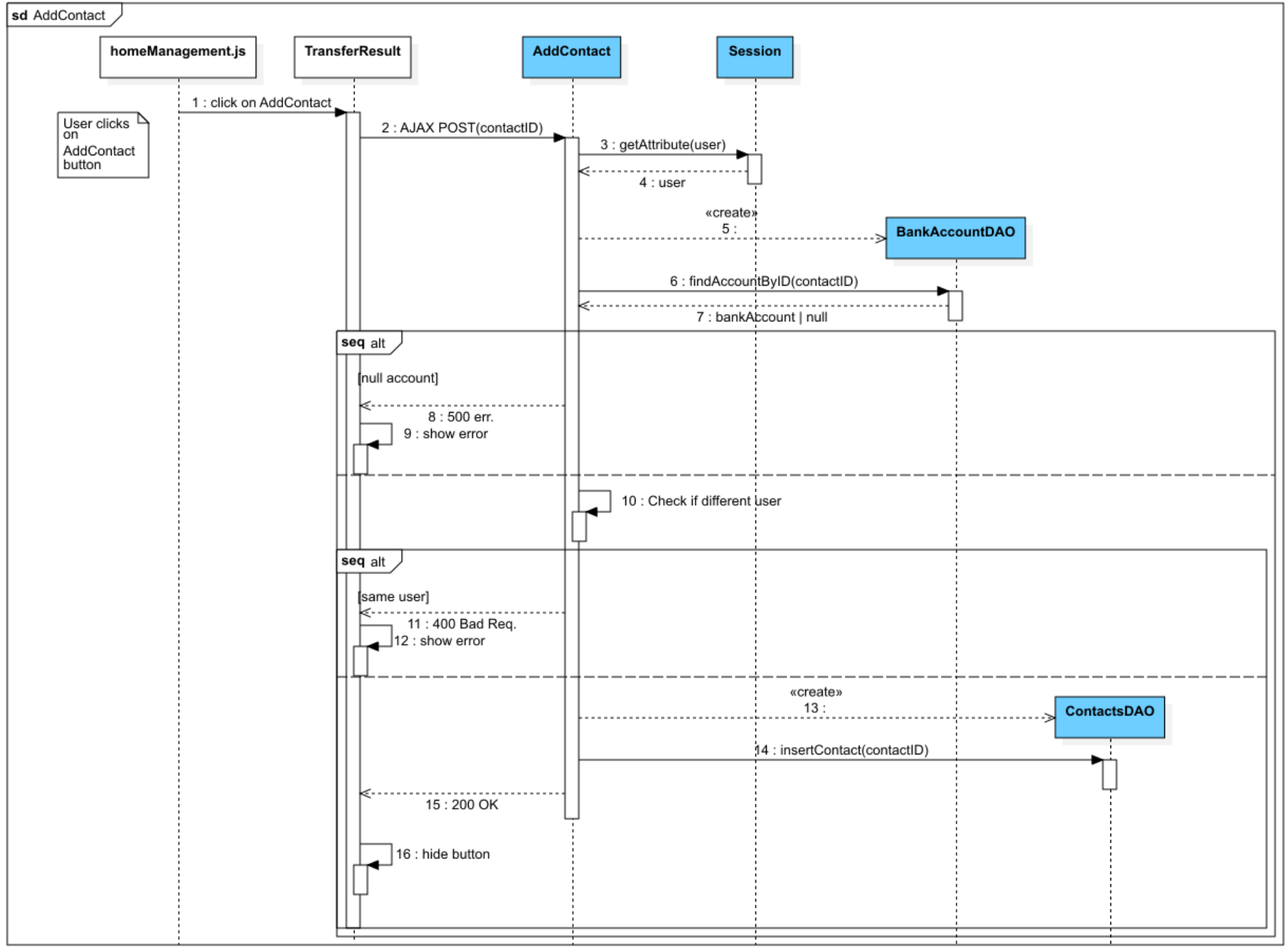

sd Submit Login
Login.html & login Management.js
Login
session
sessionStorage
window
1 : AJAX POST
email,
password
click
submit
button
alt
[null check params not OK]
2 : status 400
3 : show error
[else]
«create»
4 :
UserDAO
5 : findUser(email, password)
6 : user || null
alt
ID,
name,
surname
[user != null]
7 : status 200
8 : setAttribute("user", user)
9 : set userID, name, surname
10 : location.href = "Home.html"
[else]
11 : status 401
12 : show error

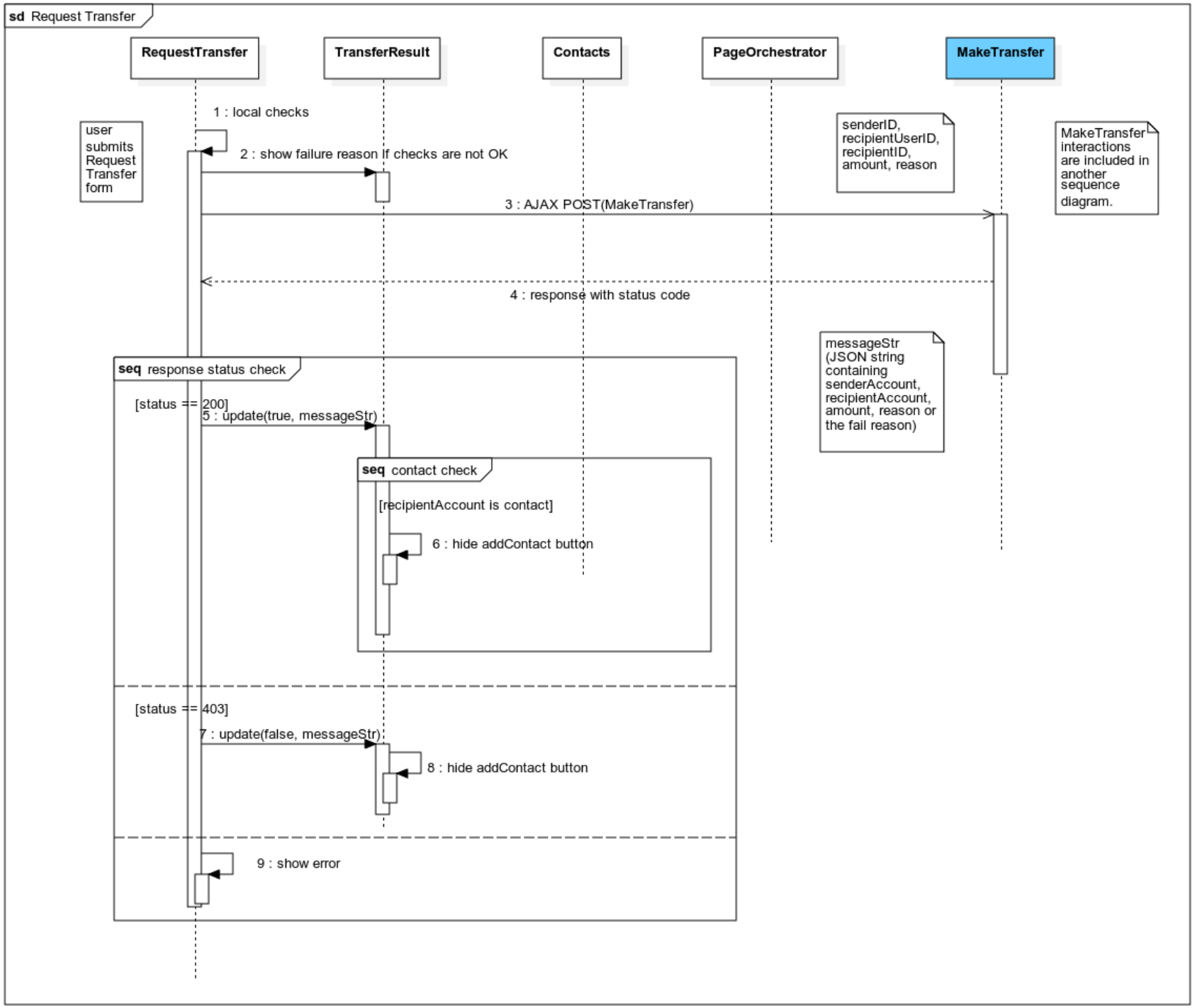

sd Logout
LogoutButton
Logout
session
sessionStorage
window
1 : AJAX GET
2 : invalidate
clicks on Logout button
3 : 200 OK
4 : removeItem("id")
5 : removeItem("name")
6 : removeItem("surname")
7 : location.href = "Login.html"

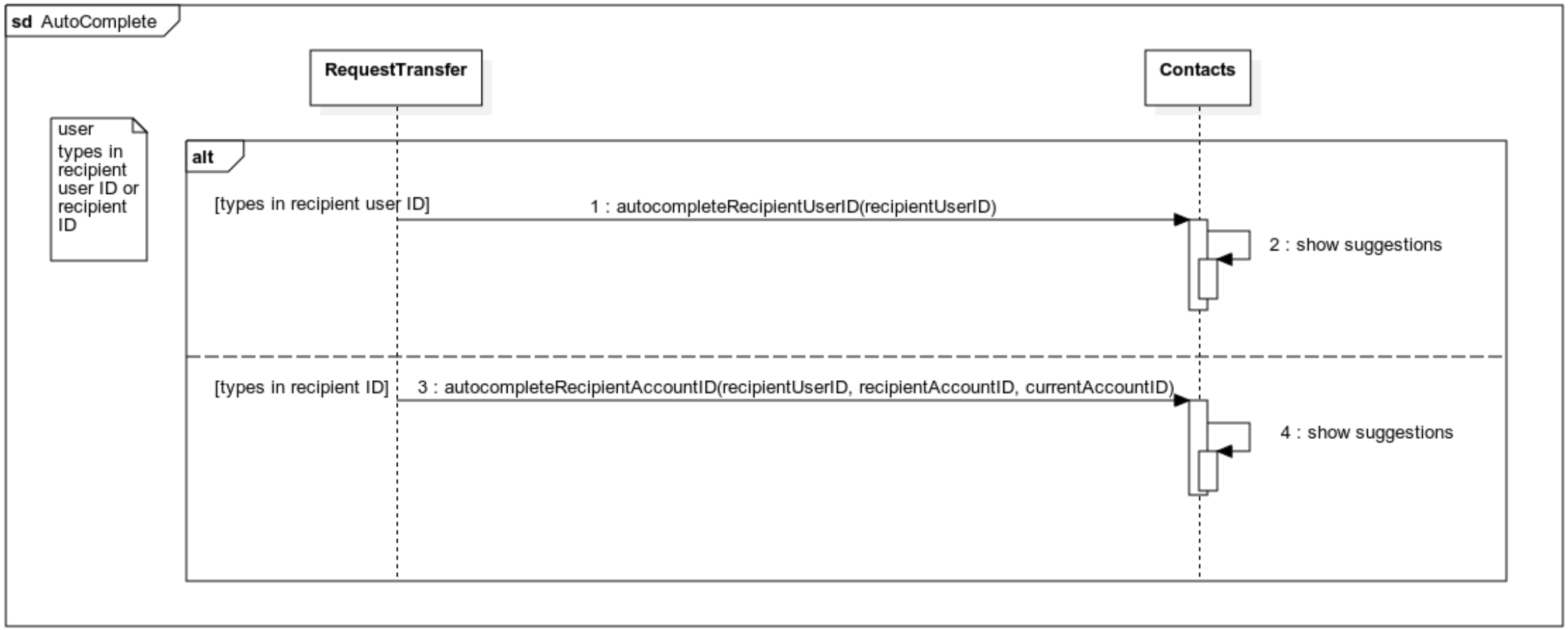

sd Homepage onLoad
homeManagement.js
SessionStorage
GetAccounts
GetContacts
«create»
1 : PageOrchestrator
Page Orchestrator
2 : Start
3 : Get UserID, Name, Surname
4 :
«create»
5 : -
UserDetails
«create»
6 : -
AccountList
«create»
7 : registerEvents
AddAccount
«create»
8 : -
AccountDetails
«create»
9 : registerEvents
TransferForm
«create»
10 : registerEvents
TransferResult
«create»
11 : load
Contacts
12 : AJAX GET
13 : load
«create»
14 : registerEvents
LogoutButton
15 : Refresh
16 : show
17 : reset
18 : reset
19 : show
20 : AJAX GET
21 : accounts


sd AddAccount (js)
homeManagement.js
AddAccount
CreateAccount
PageOrchestrator
Session
User clicks on "+" button
1 : click event
2 : show account creation form
User clicks on "Submit" button
3 : click event
4 : check form validity
5 : AJAX POST
6 : getAttribute(user)
7 : user
seq alt
[invalid input]
8 : 400 Bad Req.
9 : show error
«create»
10 :
BankAccountDAO
11 : isNameTaken(accountName)
12 : taken
seq alt
[duplicate name]
13 : error
14 : show error
15 : createAccount
16 : 200 OK
17 : refresh(accountDetails.currentAccount)

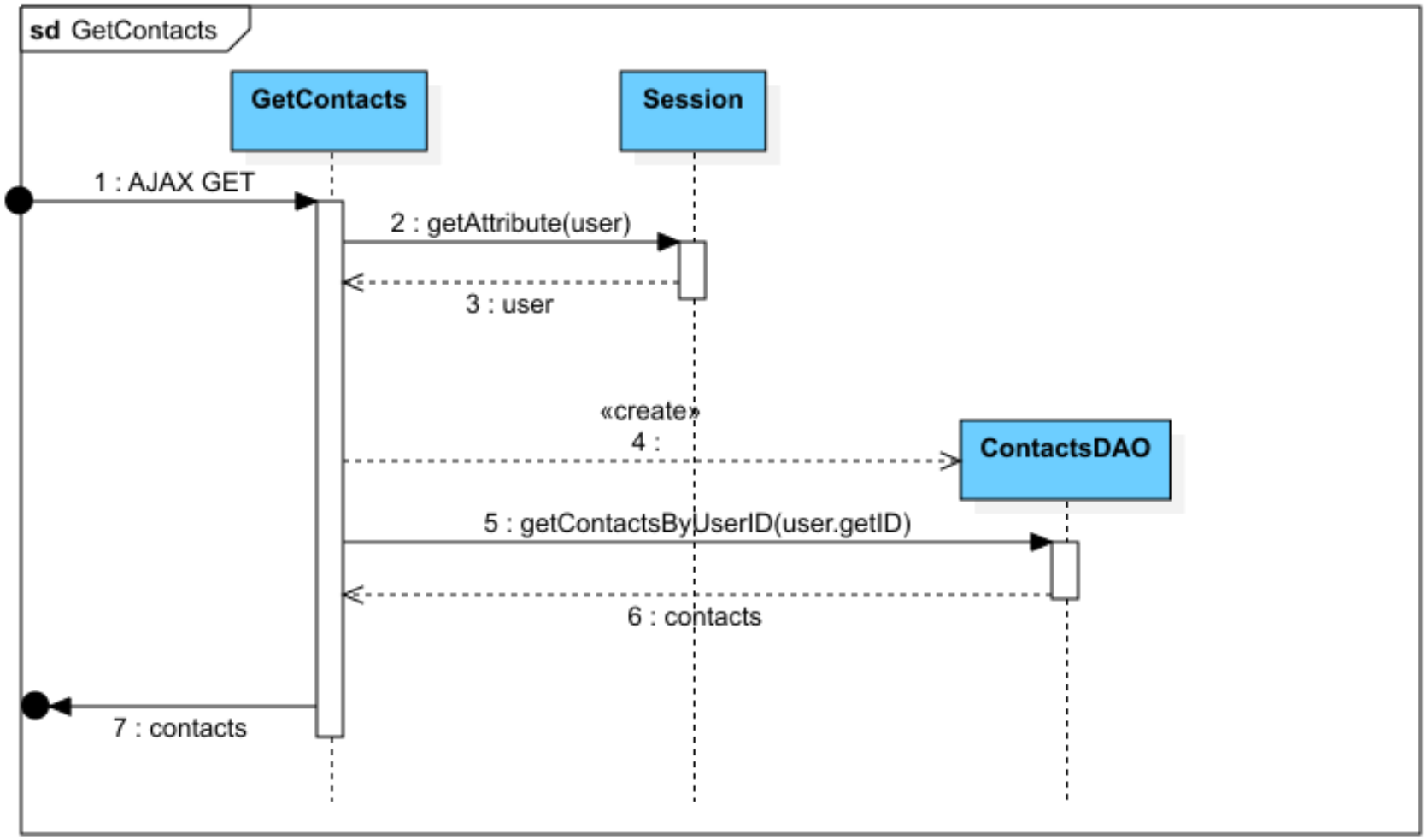

sd SelectAccount
homeManagement.js
AccountDetails
SelectAccount
Session
1 : show(accountID)
User clicks on an account from the list
2 : AJAX GET
3 : getAttribute(user)
4 : user
seq alt
«create»
5 :
BankAccountDAO
[null user]
6 : 400 Bad Req.
7 : show error
8 : findAccountByID(accountID)
9 : bankAccount
seq alt
[bankAccount == null || not the owner]
10 : 400 Bad Req.
11 : show error
«create»
12 :
TransferDAO
13 : getTransfersByAccountID(accountID)
14 : transfers
15 : bankAccount, transfers
16 : show data


sd AddContact
homeManagement.js
TransferResult
AddContact
Session
1 : click on AddContact
User clicks on AddContact button
2 : AJAX POST(contactID)
3 : getAttribute(user)
4 : user
«create»
5 :
BankAccountDAO
6 : findAccountByID(contactID)
7 : bankAccount | null
seq alt
[null account]
8 : 500 err.
9 : show error
10 : Check if different user
seq alt
[same user]
11 : 400 Bad Req.
12 : show error
«create»
13 :
ContactsDAO
14 : insertContact(contactID)
15 : 200 OK
16 : hide button

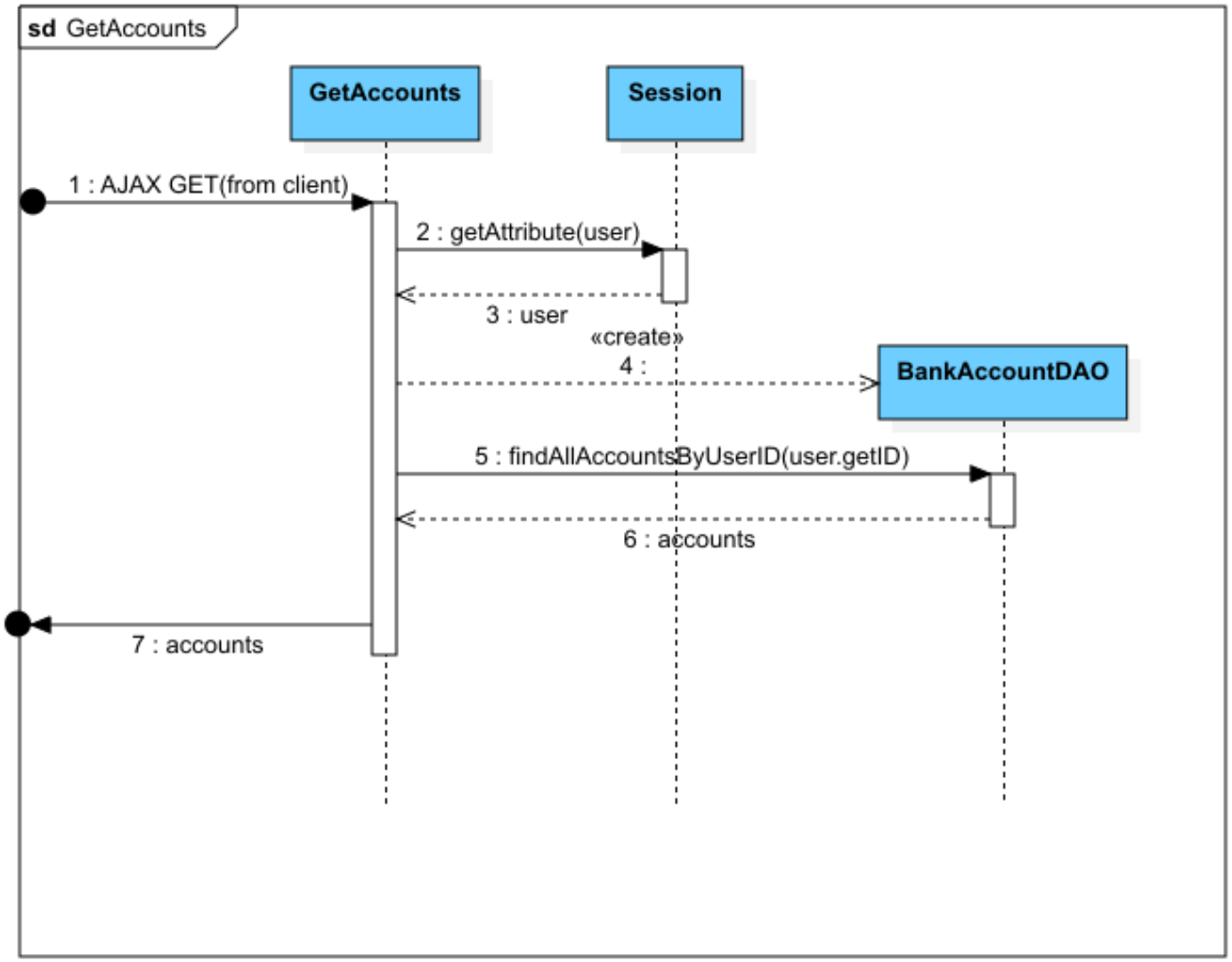

sd Request Transfer
RequestTransfer
TransferResult
Contacts
PageOrchestrator
MakeTransfer
1 : local checks
user submits Request Transfer form
2 : show failure reason if checks are not OK
senderID, recipientUserID, recipientID, amount, reason
MakeTransfer interactions are included in another sequence diagram.
3 : AJAX POST(MakeTransfer)
4 : response with status code
messageStr (JSON string containing senderAccount, recipientAccount, amount, reason or the fail reason)
seq response status check
[status == 200]
5 : update(true, messageStr)
seq contact check
[recipientAccount is contact]
6 : hide addContact button
[status == 403]
7 : update(false, messageStr)
8 : hide addContact button
9 : show error


sd AutoComplete
RequestTransfer
Contacts
user types in recipient user ID or recipient ID
alt
[types in recipient user ID]
1 : autocompleteRecipientUserID(recipientUserID)
2 : show suggestions
[types in recipient ID]
3 : autocompleteRecipientAccountID(recipientUserID, recipientAccountID, currentAccountID)
4 : show suggestions

Back to HomePage
onLoad interaction



Back to HomePage
onLoad interaction



Back to RequestTransfer interaction