Progettazione del progetto d’esame AA 2019/2020

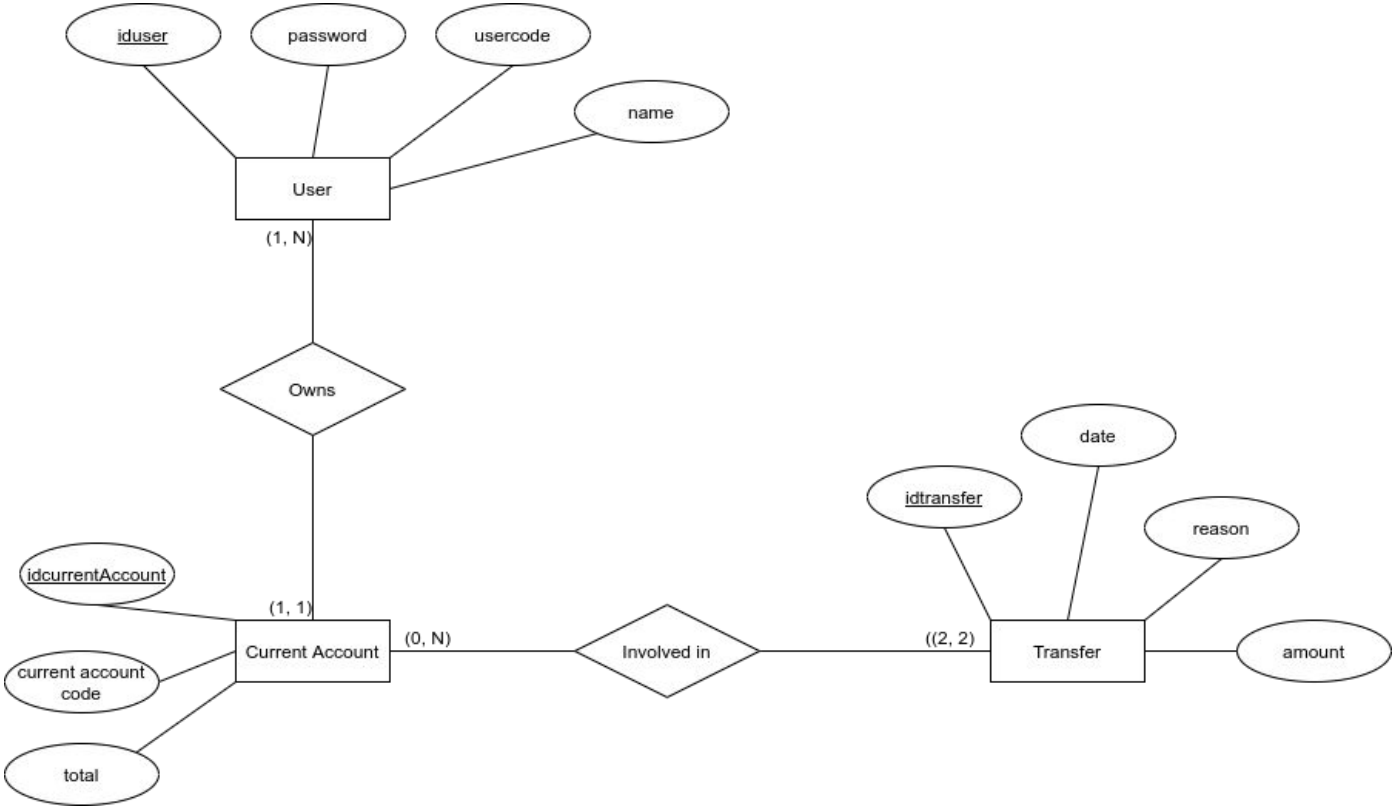
Duilio Cirino, Lorenzo Cocchia

Scelte architetturali

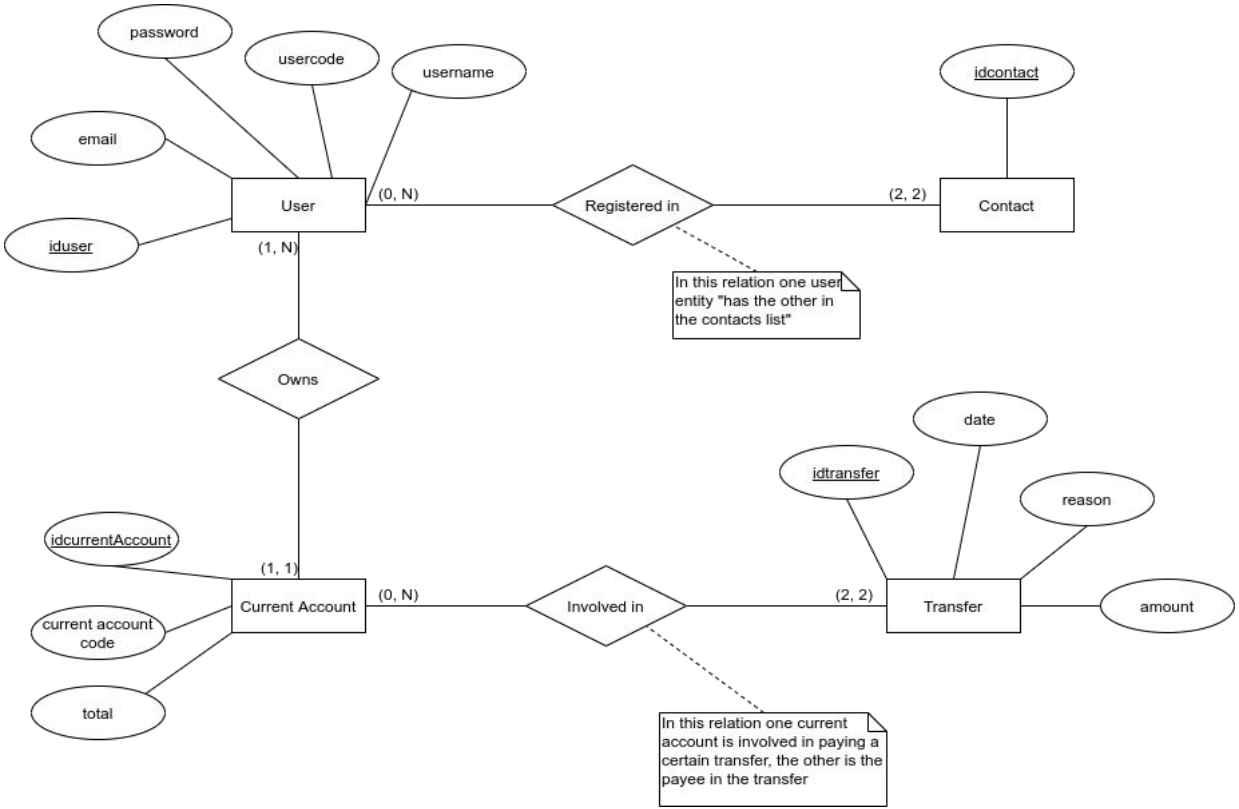
Da specifica, devono distinguersi due progetti. Uno con una architettura tipica delle applicazioni pure HTML e l’altro secondo l’architettura RIA

Analisi delle specifiche

Analisi dei dati, modello concettuale HTML p.

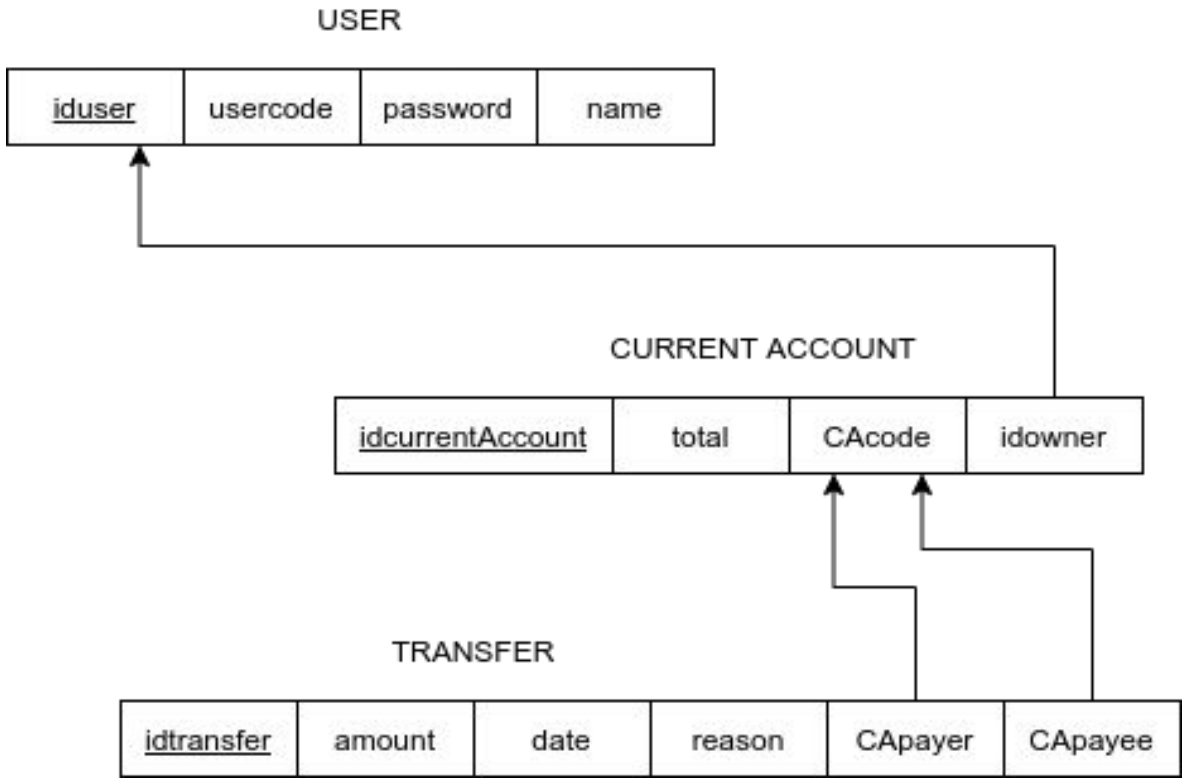


Analisi dei dati, modello concettuale RIA

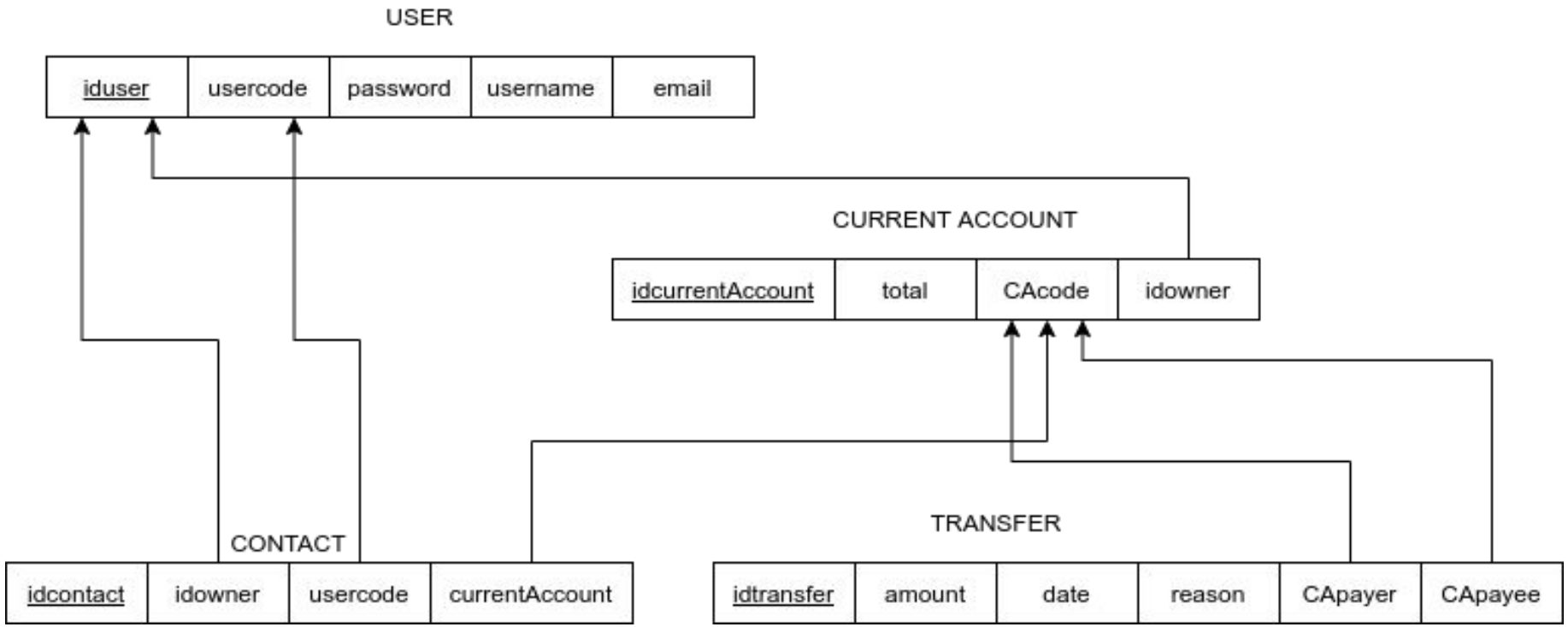


Note:
i) L'entità "Contact" è un ausilio che si è deciso di utilizzare per chiarezza per non utilizzare una auto-relazione molti-a-molti in "User".
ii) Anche l'entità transfer può essere vista come "tabella ponte" di una auto-relazione molti-a-molti in "Current Account"

Analisi dei dati, modello logico HTML pure



Analisi dei dati, modello logico RIA



Scelte architetturali per la base di dati

Si è deciso di usare come foreign keys i codici che devono essere unici all'interno delle loro tabelle. Questi codici (con attributo UNIQUE in SQL) possono essere considerati come chiavi: al contempo però non si è voluto rinunciare alla comodità di avere una chiave intera all'interno delle tabelle della base di dati.

Viste

HTML pure:

1. Pagina di login
2. Homepage
3. Pagina di stato del conto corrente
4. Pagina di ricevuta con successo del pagamento
5. Pagina di ricevuta con fallimento del pagamento

RIA:

1. Pagina di login
2. Pagina di registrazione
3. Homepage. Da specifica è richiesto che dopo che l'utente ha effettuato con successo il login l'applicazione deve essere contenuta in una pagina

N.B.: Nelle specifiche dell'applicazione RIA si fa riferimento alla possibilità che l'utente si possa registrare

Componenti delle viste

HTML pure:

1. Login: form di login
2. Homepage: lista dei conti corrente dell'utente
3. Stato CC:
   a. Dettagli del CC
   b. Lista di tutti i trasferimenti effettuati
   c. Form per l'avvio di un nuovo trasferimento di denaro
4. Pagamento avvenuto/Pagamento fallito: link per continuare ad utilizzare l'applicazione.

RIA:

1. Login: come HTML pure
2. Registrazione: form per la sottomissione dei dati per potersi registrare
3. Homepage(non necessariamente tutte in modo simultaneo):
   a. lista dei conti correnti
   b. lista dei trasferimenti fatti da un certo conto corrente
   c. Form per la sottomissione di un nuovo trasferimento
   d. componente per il messaggio personale

Eventi

1. L’utente accede con le proprie credenziali sottomettendo il form di login
2. L’utente accede alla homepage ([RIA] la homepage si carica)
3. L’utente seleziona un conto dalla lista dei propri conti
4. L’utente sottomette il form per la creazione di un nuovo trasferimento
5. L’utente decide di tornare alla lista dei conti correnti dopo aver ricevuto la risposta del nuovo trasferimento
6. L’utente decide di tornare ai dettagli del conto corrente dopo aver ricevuto la risposta del nuovo trasferimento
7. (solo RIA) l’utente si registra sottomettendo il form di registrazione

N.B. Nel caso RIA dato che deve essere tutto contenuto in una pagina l’evento 6 non diventa un evento dove l’utente ha arbitrio

Azioni

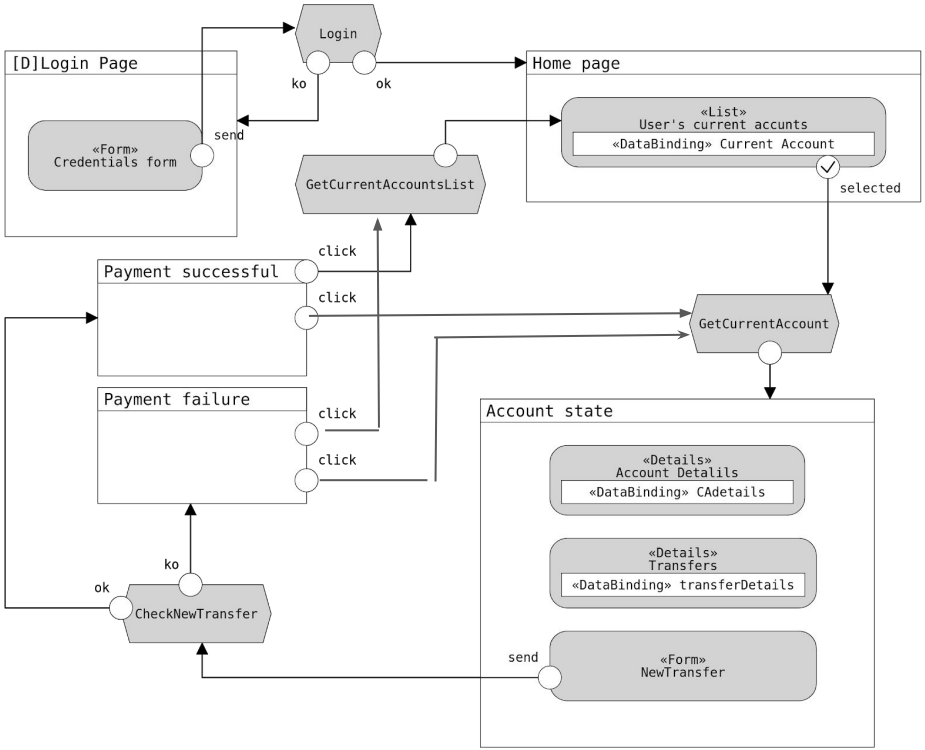
1. Accesso
2. Controllo dati credenziali
3. (solo RIA) Registrazione
4. (solo RIA) Controllo dati registrazione
5. Creazione di un nuovo trasferimento di denaro
6. Controllo di fattibilità del trasferimento
7. Log out

Requisiti non funzionali e completamento della specifica

- Da specifica, per l'architettura RIA il controllo dati deve essere sia client che server-side.
- Per l'architettura RIA, la rubrica salvata di ogni utente viene spedita al client non appena il cliente esegue il login con successo. In caso di aggiunta nel corso della sessione, essa dovrà essere aggiornata.
- Per poter mantenere il vincolo di atomicità si è deciso di sfruttare le transazioni di SQL (e quindi anche l'isolamento delle operazioni secondo le proprietà ACID di una base di dati).
- Nonostante non fosse esplicito nella specifica abbiamo assunto che una volta che l'utente effettua l'accesso può sempre effettuare il Logout.
- Per semplicità, e anche per potersi avvicinare maggiormente al caso reale, si è deciso di usare come codici utente e come codice di conto corrente delle stringhe numeriche di 4 caratteri.

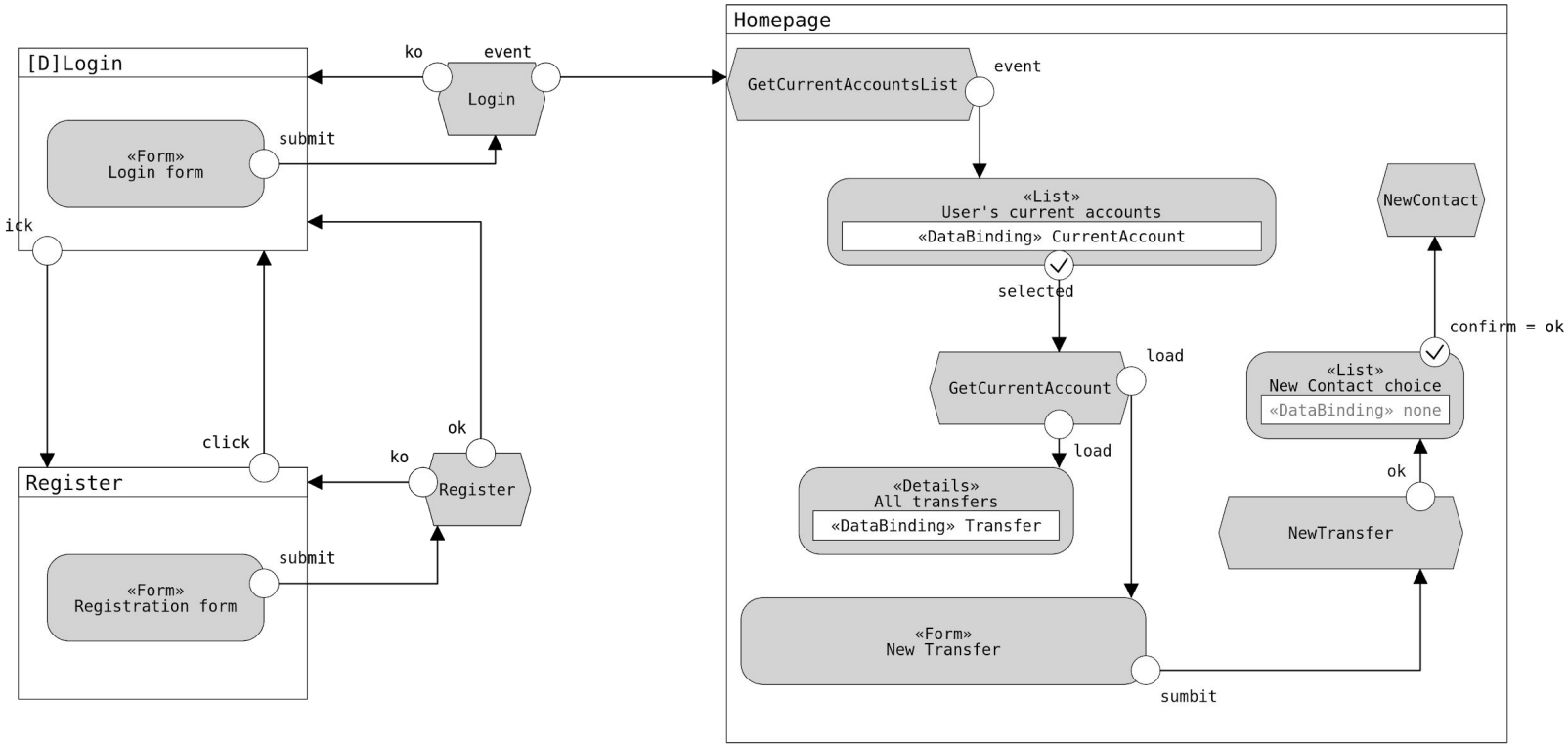
Design dell’applicazione

Diagramma IFML HTML pure



- per semplicità non è stata disegnata l'azione di Logout

Diagramma IFML RIA



Corrispondenza evento-azione HTML pure

Il diagramma IFML è autoesplicativo in questo caso.

Corrispondenza evento-azione RIA

Client-side (evento -> azione)	Server-side (evento -> azione)
sottomissione form login -> invio richiesta controllo credenziali	post(username, password) -> invocazione servlet Login
sottomissione form registrazione -> controllo validità credenziali	----------
esito positivo controllo credenziali -> invio richiesta di controllo di unicità dello username	get(username) -> invocazione servlet CheckUsername
esito positivo unicità username -> invio richiesta di registrazione	post(username, password, repeated password, email -> invocazione servlet Registration
esito negativo registrazione -> stampa a display dell'errore ricevuto	----------

Corrispondenza evento-azione RIA

esito positivo registrazione -> reindirizzazione alla pagina di login	----------
responso positivo all'accesso con credenziali -> reindirizzazione alla homepage	----------
successo del caricamento del documento -> mostra messaggio personale	----------
successo del caricamento del documento -> invio richiesta di possesso di conti correnti	get( ) -> invocazione servlet GetCurrentAccountsList
successo del caricamento dei conti correnti -> invio richiesta contatti dell'utente	get( ) -> invocazione servlet GetAllContacts
successo del caricamento dei conti correnti -> autoclick che clicca il primo elemento della lista	----------

Corrispondenza evento-azione

click su un elemento della lista di CC -> invio richiesta dei trasferimenti relativi al CC cliccato	get(idCurrentAccount) -> invocazione servlet GetAllTransfers
successo del caricamento dei trasferimenti -> creazione della tabella che li possa mostrare	----------
successo del caricamento dei trasferimenti -> creazione del form per la sottomissione di un nuovo trasferimento	----------
inserimento nel campo codice utente -> controllo di presenza nei contatti di un codice utente che corrisponda alla stringa inserita	----------
responso positivo al controllo di presenza nei contatti -> auto-inserimento del codice persona e del codice CC presente nei contatti	----------

Corrispondenza evento-azione

sottomissione del form -> controllo validità dei campi	----------
responso positivo dal controllo di validità della form -> invio dei dati per un nuovo trasferimento	post(amount, reason, current account code payer (hidden), current account code payee, user code payee) -> invocazione servlet NewTransfer (con controllo di fattibilità dell'operazione annesso)
successo del nuovo trasferimento -> ricaricamento della lista dei CC	get( ) -> invocazione servlet GetCurrentAccountsList
… (stesso procedimento fatto per il caricamento iniziale della pagina)	

Server: JavaBeans e DAO (HTML pure)

1. CurrentAccountBean
2. TransferBean
3. UserBean

1. CurrentAccountDAO
   a. getCAByCode(String CAcode): CurrentAccountBean
   b. getCAById(int idcurrentAccount): CurrentAccountBean
   c. getCAsByUser(int iduser): List<CurrentAccountBean>
   d. getCheckByCode(String CAcode) : float
   e. updateCheckByAmount(float amount, String CAcode): boolean
2. TransferDAO
   a. getTransferByCAId(int idCA): List<TransferBean>
   b. newTransfer(float amount,String reason, String CApayer, String CApayee): boolean
3. UserDAO
   a. checkCredentials(String personalCode, String password): UserBean
   b. getUserByCode(String usercode): UserBean

Server: JavaBeans e DAO (RIA)

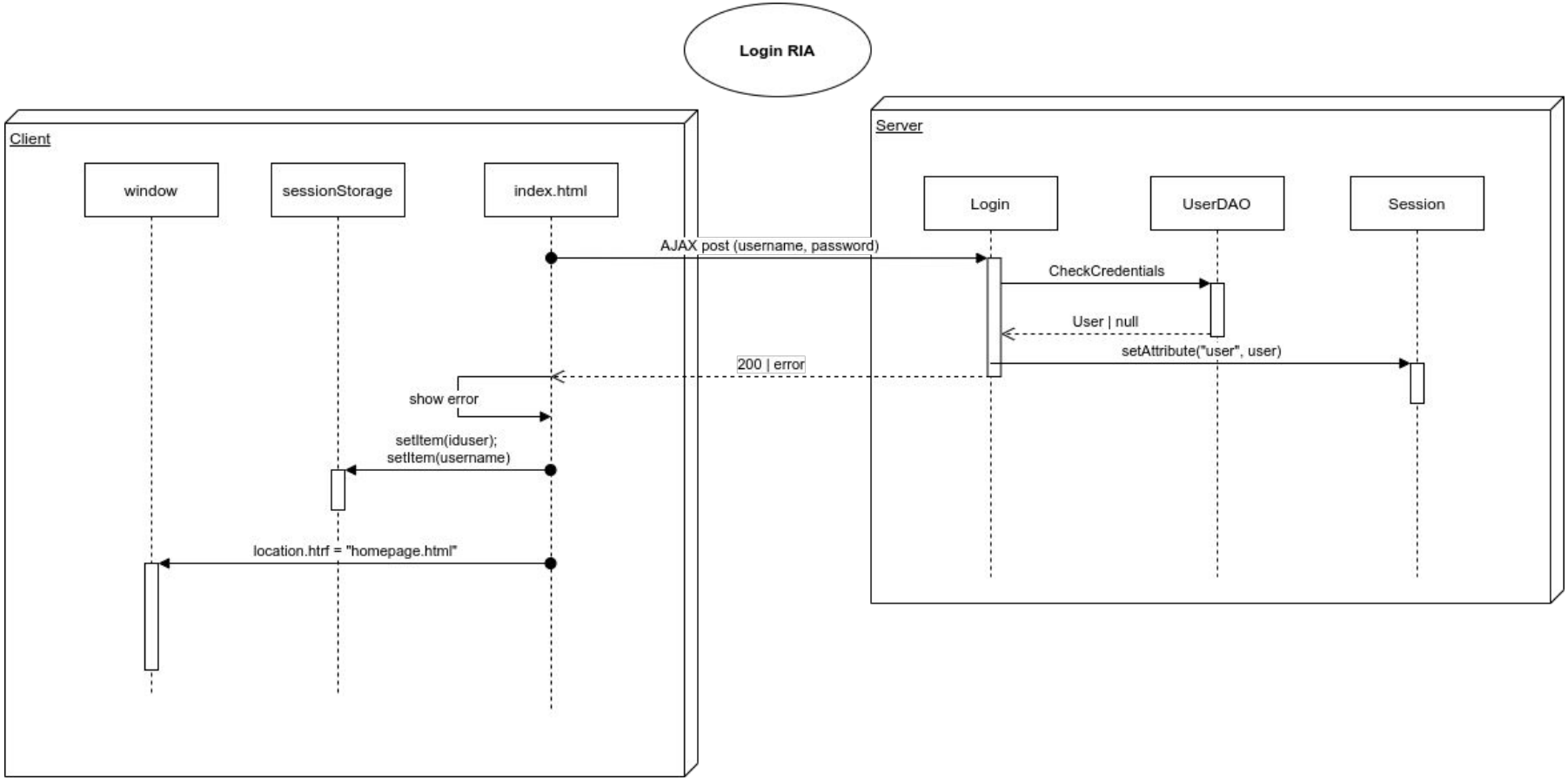
1. Contact
2. CurrentAccount
3. Transfer
4. User

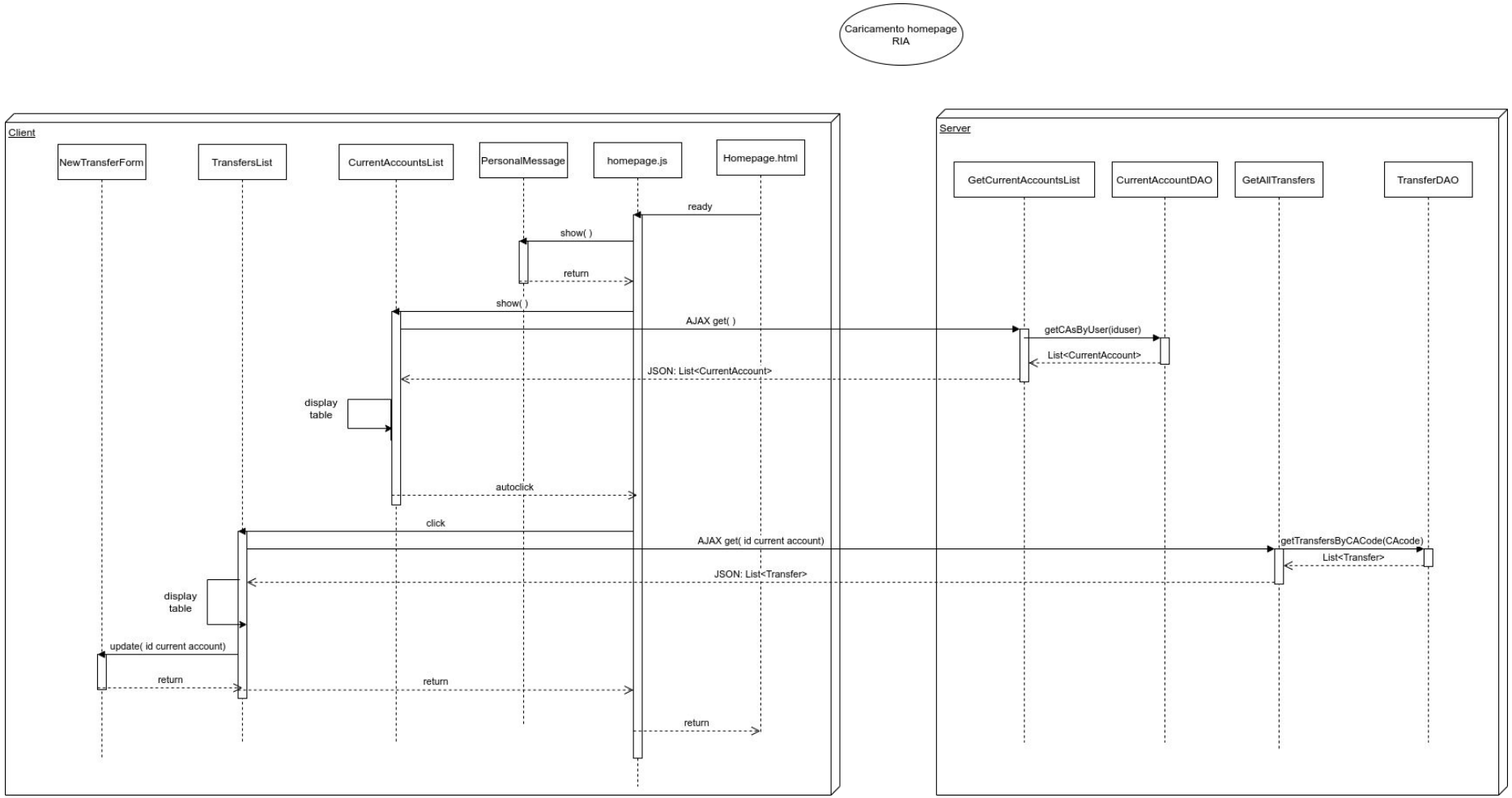
1. ContactDAO
    a. newUsersContact(int iduser, String CAcodeContact) : boolean
    b. getAllUsersContacts(int idUser) : List<Contact>
2. CurrentAccountDAO
    a. getCAByCode(String CAcode) : CurrentAccount
    b. getCAById(int idcurrentAccount) : CurrentAccount
    c. getCAsByUser(int iduser) : List<CurrentAccount>
    d. getCheckByCode(String CAcode) : float
    e. updateCheckByAmount(float amount, String CAcode) : boolean
3. TransferDAO
    a. getTransferByCAId(String CAcode): List<Transfer>
    b. newTransfer(float amount,String reason, String CApayer, String CApayee): boolean
4. UserDAO
    a. checkCredentials(String username, String password) : User
    b. newUser(String username, String password, String email) : boolean
    c. getUserByCode(String CAcode): User
    d. getIserById(int iduser): User
    e. usernameIsYetExistent(String username): boolean

Client: componenti delle viste (RIA)

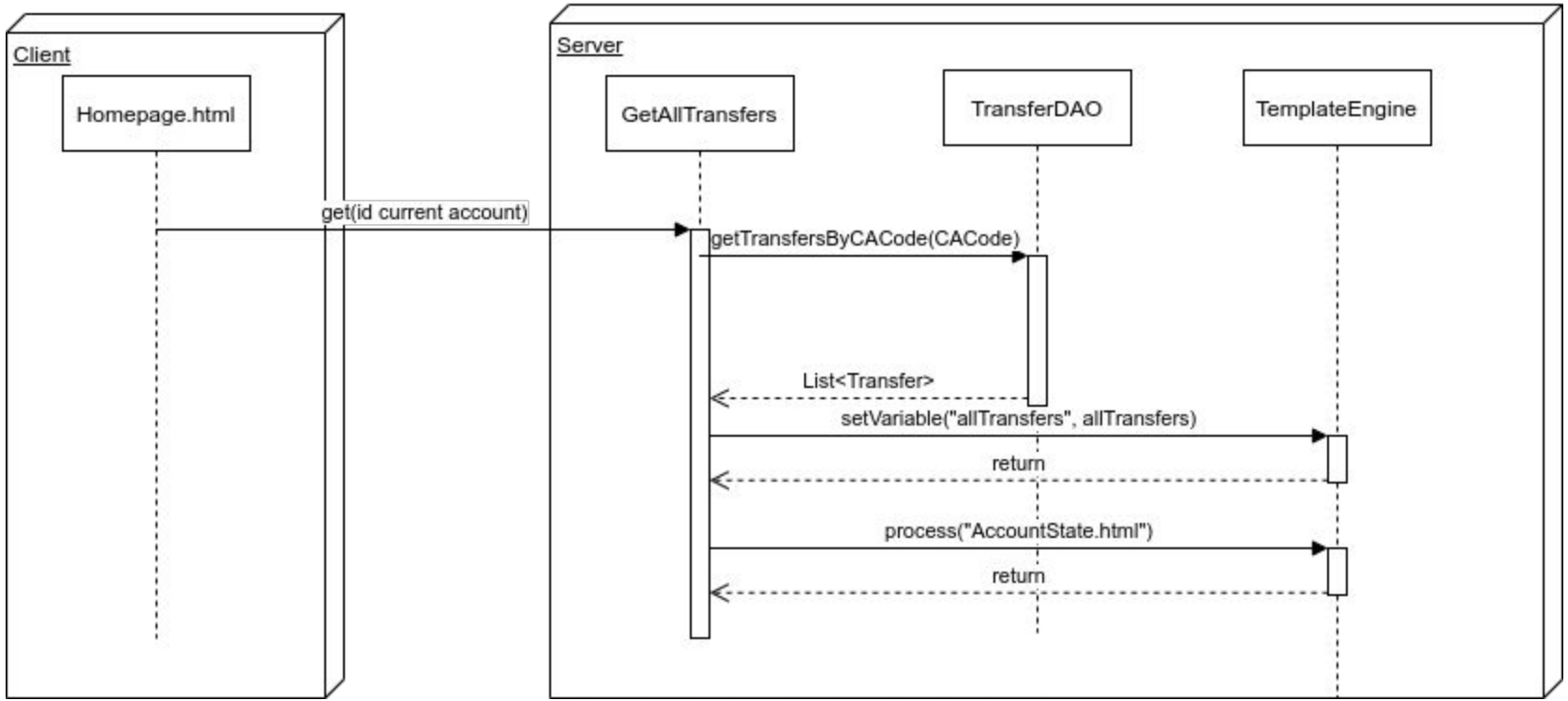
1. PersonalMessage
    a. username
    b. show( )
2. CurrentAccountsList
    a. show( )
    b. update( )
    c. autoclick( )
3. TransfersList
    a. show( )
    b. highlightCurrentAccount( )
4. NewTransferForm
    a. submitRegister
    b. contactTracker
    c. update( )
5. Contacts
    a. contacts
    b. getContacts( )
    c. retrieveContacts( )
    d. addContact( )

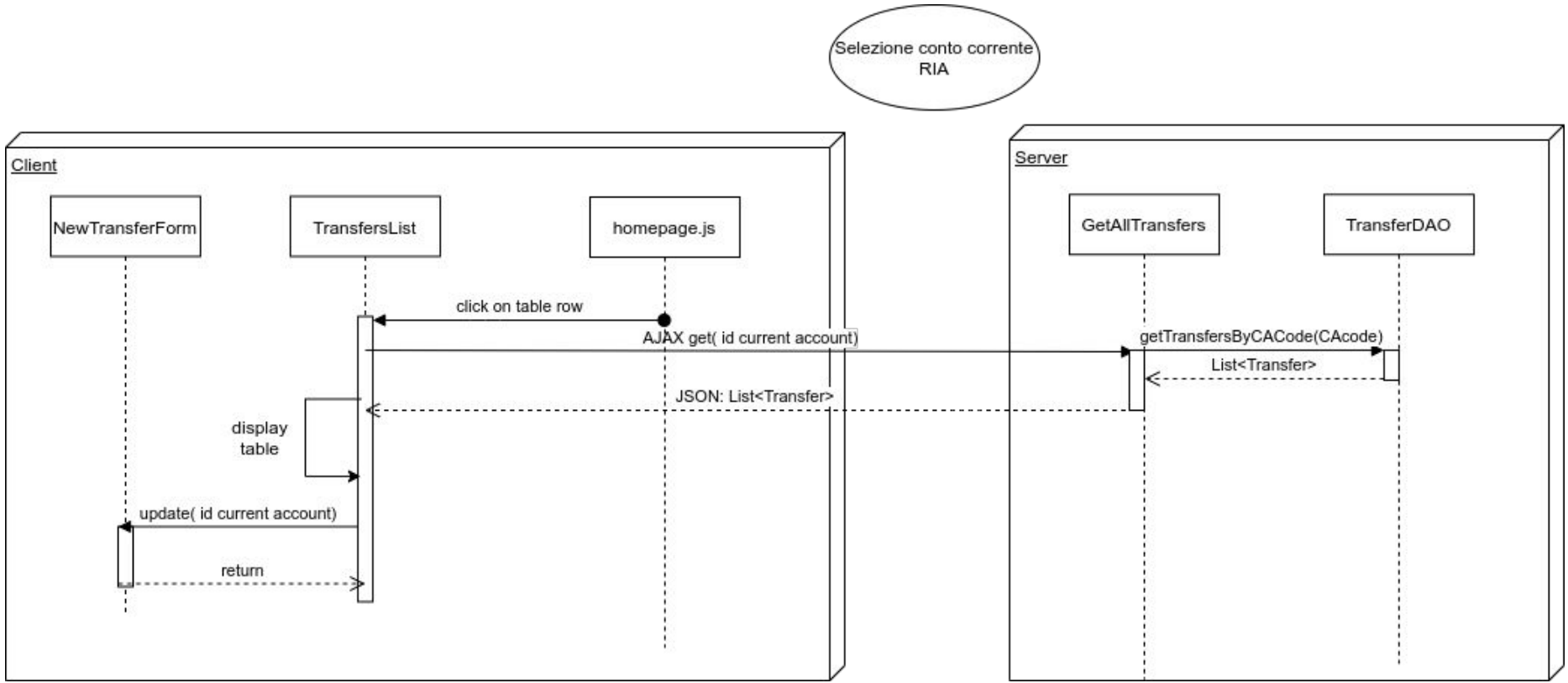
Casi d’uso

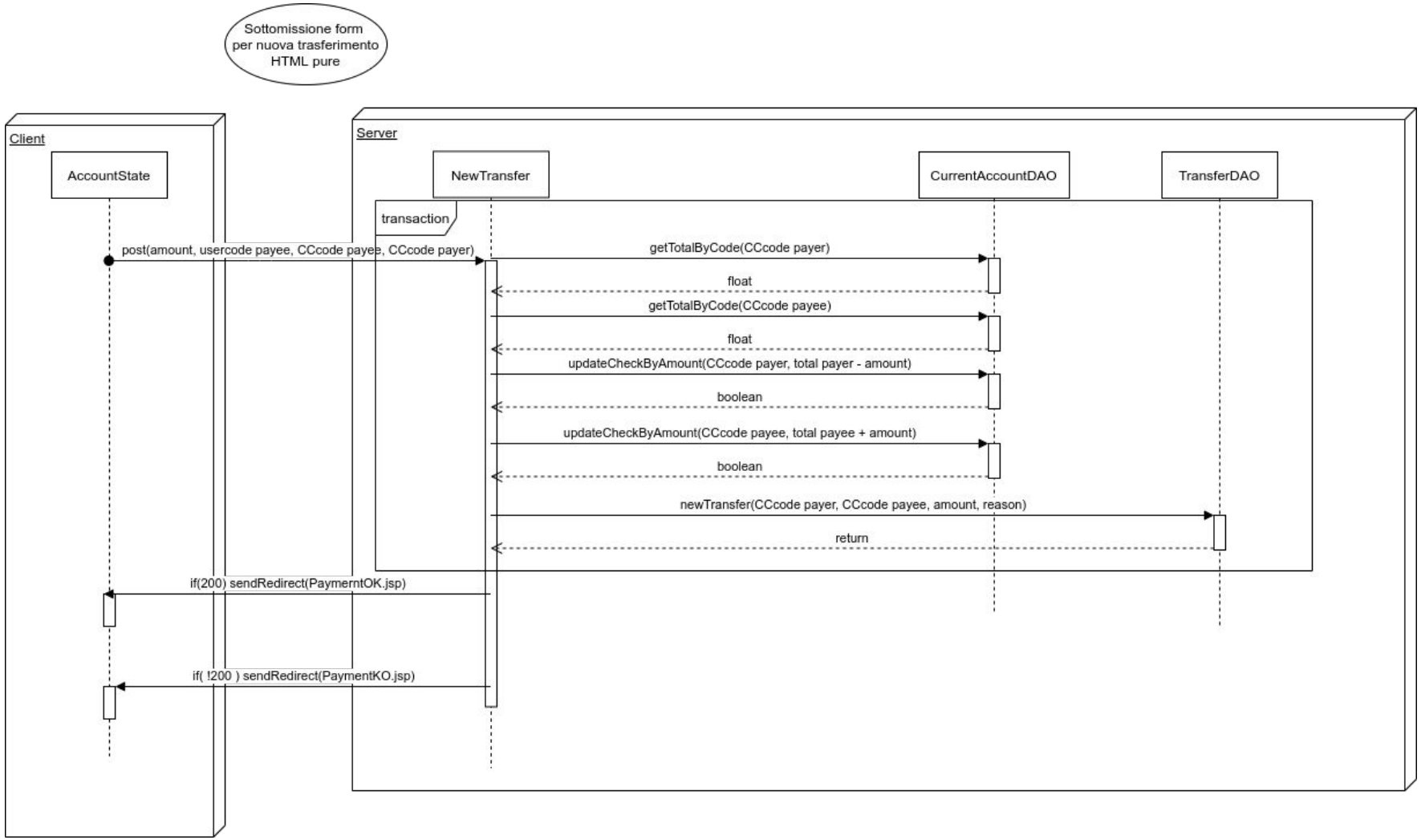

Login RIA
Client
window
sessionStorage
index.html
Server
Login
UserDAO
Session
AJAX post (username, password)
CheckCredentials
User | null
setAttribute("user", user)
200 | error
show error
setItem(iduser);
setItem(username)
location.htrf = "homepage.html"

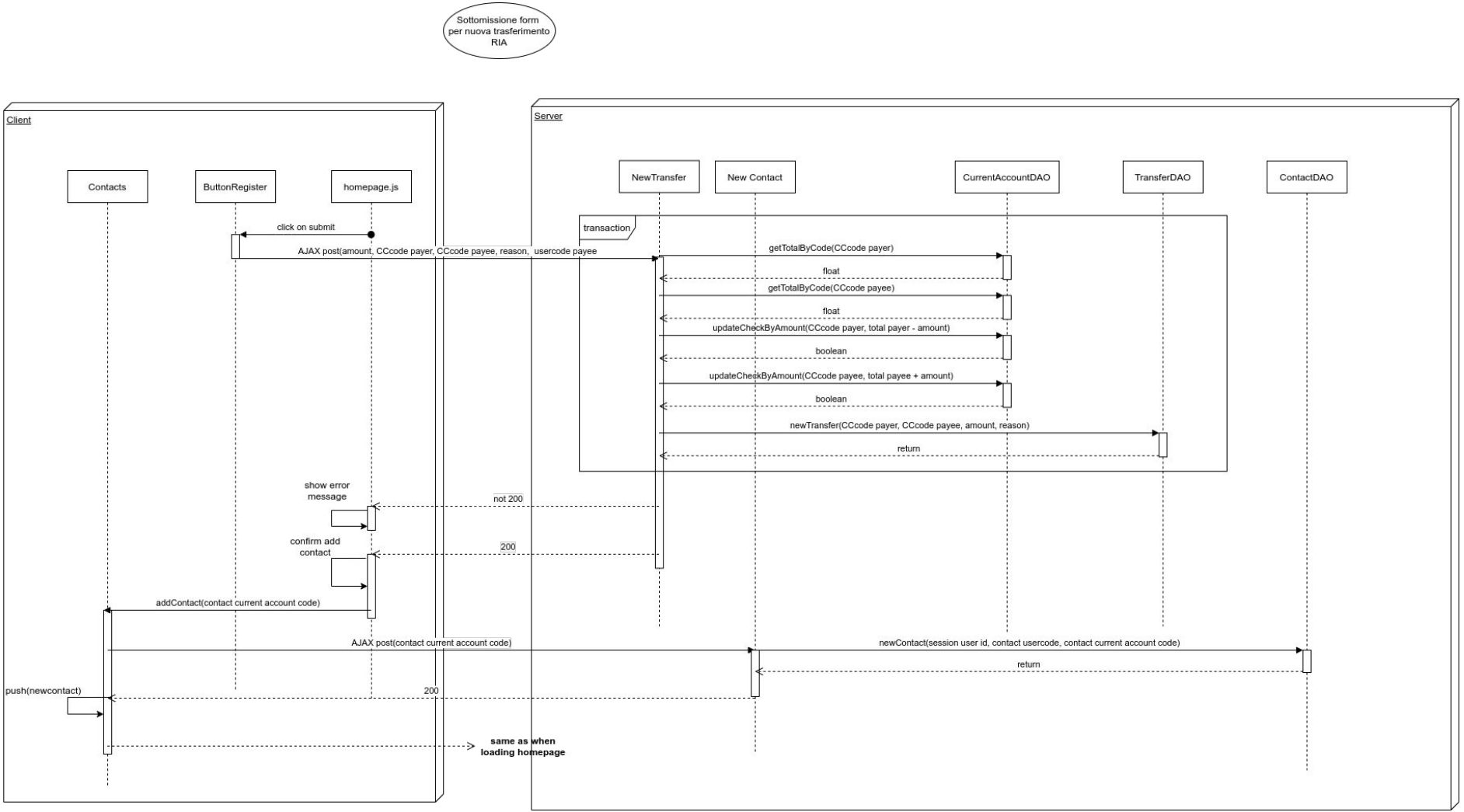

Caricamento homepage RIA
Client
Server
NewTransferForm
TransfersList
CurrentAccountsList
PersonalMessage
homepage.js
Homepage.html
GetCurrentAccountsList
CurrentAccountDAO
GetAllTransfers
TransferDAO
ready
show( )
return
show( )
AJAX get( )
getCAsByUser(iduser)
List<CurrentAccount>
JSON: List<CurrentAccount>
display table
autoclick
click
AJAX get(id current account)
getTransfersByCACode(CAcode)
List<Transfer>
JSON: List<Transfer>
display table
update(id current account)
return
return
return

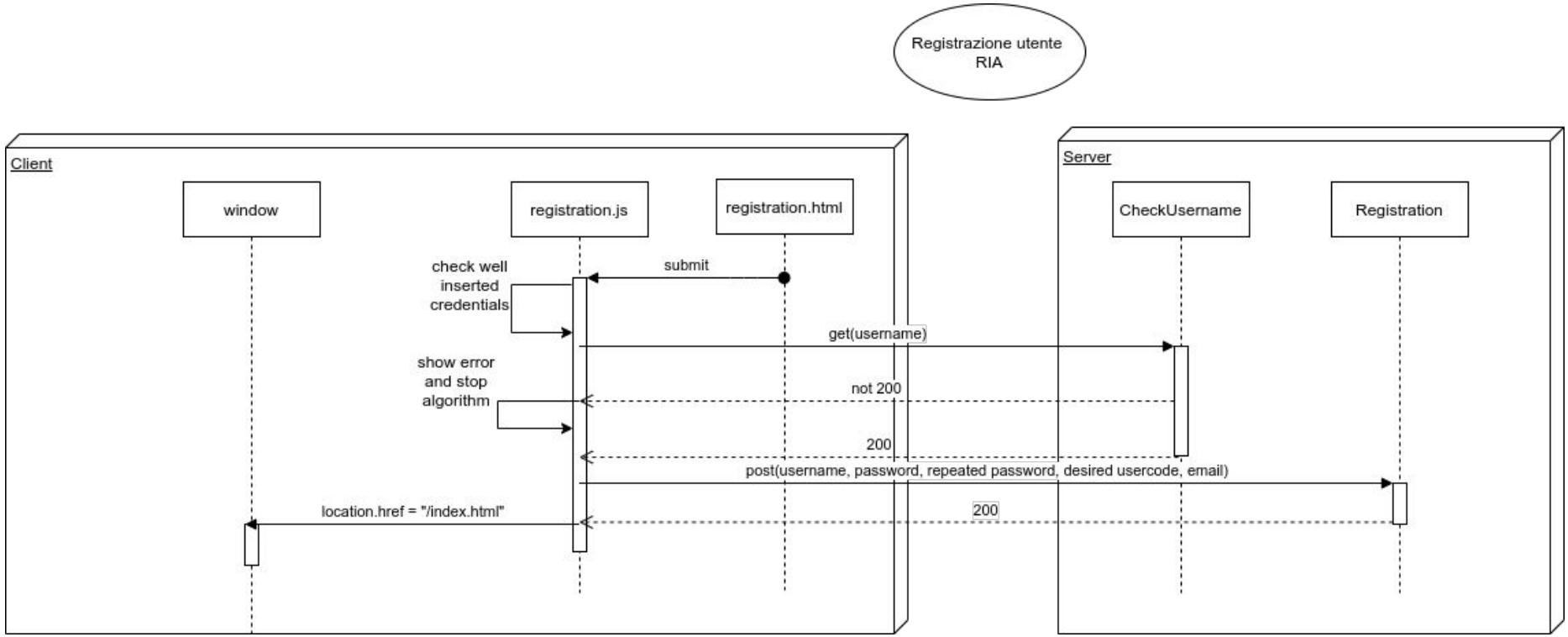
Selezione conto corrente
HTML pure




Selezione conto corrente
RIA
Client
NewTransferForm
TransfersList
homepage.js
click on table row
AJAX get(id current account)
display
table
update(id current account)
return
Server
GetAllTransfers
TransferDAO
getTransfersByCACode(CAcode)
List<Transfer>
JSON: List<Transfer>


Sottomissione form
per nuova trasferimento
HTML pure
Client
Server
AccountState
NewTransfer
CurrentAccountDAO
TransferDAO
transaction
post(amount, usercode payee, CCcode payee, CCcode payer)
getTotalByCode(CCcode payer)
float
getTotalByCode(CCcode payee)
float
updateCheckByAmount(CCcode payer, total payer - amount)
boolean
updateCheckByAmount(CCcode payee, total payee + amount)
boolean
newTransfer(CCcode payer, CCcode payee, amount, reason)
return
if(200) sendRedirect(PaymerntOK.jsp)
if(!200) sendRedirect(PaymentKO.jsp)


Sottomissione form per nuova trasferimento RIA
Client
Server
Contacts
ButtonRegister
homepage.js
NewTransfer
New Contact
CurrentAccountDAO
TransferDAO
ContactDAO
transaction
click on submit
AJAX post(amount, CCcode payer, CCcode payee, reason, usercode payee
getTotalByCode(CCcode payer)
float
getTotalByCode(CCcode payee)
float
updateCheckByAmount(CCcode payer, total payer - amount)
boolean
updateCheckByAmount(CCcode payee, total payee + amount)
boolean
newTransfer(CCcode payer, CCcode payee, amount, reason)
return
show error message
not 200
confirm add contact
200
addContact(contact current account code)
AJAX post(contact current account code)
newContact(session user id, contact usercode, contact current account code)
return
push(newcontact)
200
same as when loading homepage


Registrazione utente RIA
Client
Server
window
registration.js
registration.html
CheckUsername
Registration
check well inserted credentials
submit
get(username)
show error and stop algorithm
not 200
200
post(username, password, repeated password, desired usercode, email)
location.href = "/index.html"
200