Esercizio 4: trasferimento denaro

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.

- Entities, attributes, relationship

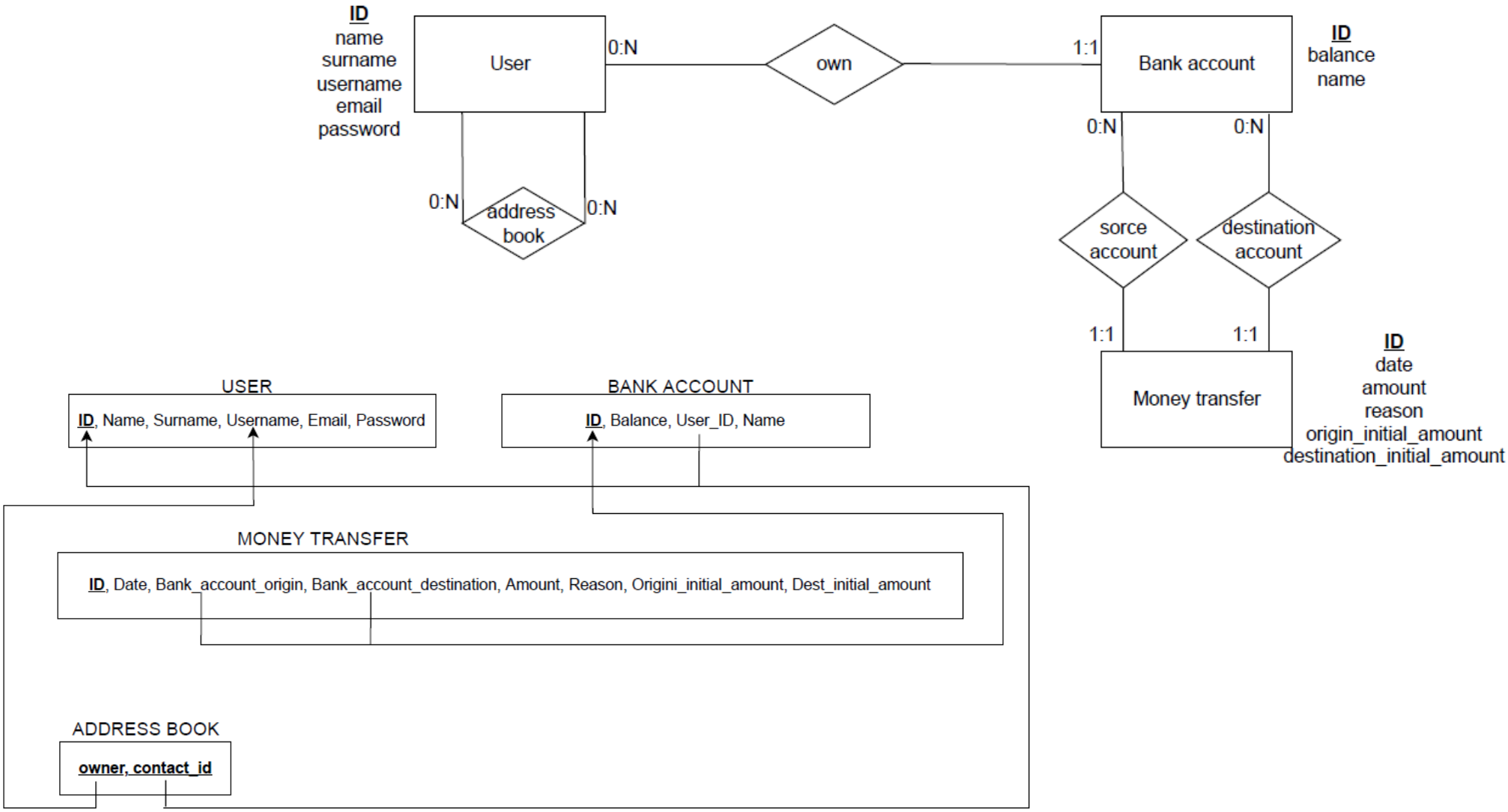
Versione RIA

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.

- Entities, attributes, relationship

Database design



Database schema (1/3)

```
CREATE TABLE 'User' (
        'id' int AUTO_INCREMENT primary key,
        'name' varchar(45) not null,
        'surname' varchar(45) not null,
        'username' varchar(45) unique not null,
        'e-mail' varchar(60) unique not null,
        'password' varchar(45) not null
)
```

```
CREATE TABLE 'BankAccount' (
        'id' int(11)  AUTO_INCREMENT primary key,
        'balance'  decimal(10, 2) not null default '0.00',
        'user_id' int not null,
        'name' varchar(45) not null,
        costraint 'userAccount' foreign key ('user_id')
                                references 'User'('id') on update cascade
                                                         on delete cascade
)
```

Database schema (2/3)

```
CREATE TABLE 'MoneyTransfer' (
        'id' int(11) AUTO_INCREMENT primary key,
        'date' timestamp not null default CURRENT_TIMESTAMP,
        'bankAccountOrigin' int not null,
        'bankAccountDestination' int not null,
        'amount' decimal(10, 2) not null,
        'reason' varchar(60) not null,
        'origin_initial_amount' decimal(10, 2) not null,
        'destination_initial_amount' decimal(10, 2) not null,
        costraint 'sourceAccount' foreign key ('bankAccountOrigin')
                                        references 'BankAccount'('id') on update cascade
                                                                        on delete no action,
        costraint 'destinationAccount' foreign key ('bankAccountDestination')
                                        references 'BankAccount'('id') on update cascade
                                                                        on delete no action
)
```

Database schema (3/3)

```
CREATE TABLE ‘AddressBook’ (
        ‘owner’ varchar(45)  not null,
        ‘contact_id’ int not null,
        primary key (owner, contact_id),
        costraint ‘ownerUser’ foreign key (‘owner’)
                                references ‘User’('name') on update cascade
                                                          on delete cascade,
        costraint ‘contact’ foreign key (‘contact_id’)
                                references ‘User’('id') on update cascade
                                                        on delete cascade
)
```

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, view components, events, actions.

Completamento delle specifiche

- A seguito del login, viene riconosciuto l'utente e mostrato un messaggio di saluto.
- L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form.
- Il controllo delle credenziali di login viene fatto su mail e password.
- Dopo una registrazione avvenuta con successo, si viene reindirizzati alla pagina che contiene il login.
- Se un utente è loggato, può fare il logout e tornare nella pagina di login.
- Se un utente cerca di accedere ad informazioni per cui non dispone l'autorizzazione viene indirizzato alla pagina dell'applicazione, se l'utente è loggato, altrimenti alla pagina di login.
- Ad ogni conto è associato un nome che è univoco per ogni utente.
- Dalla pagina che ospita l'applicazione è possibile creare un nuovo conto specificando il nome, se l'utente ha già un conto con lo stesso nome, ciò verrà notificato, altrimenti il conto verrà creato (e la pagina aggiornata in modo asincrono).
- L'avviso di conferma del trasferimento è realizzato mediante un messaggio nella pagine che ospita l'applicazione.
- Un conto non può avere saldo negativo.
- L'importo del trasferimento deve essere strettamente positivo.
- Un trasferimento non deve avvenire se il conto sorgente e destinazione sono uguali tra loro.
- A seguito del login, riconoscere l'utente e mostrare un messaggio di saluto con il suo nome.
- Durante l'inserimento del codice del destinatario, vengono suggeriti in un menù a tendina l'elenco dei destinatari in rubrica. Selezionato il destinatario vengono suggeriti in un menù a tendina l'elenco dei suoi conti.

Server-side components

Model object (*Beans*):

- User
- BankAccount
- MoneyTransfer
- AddressBook

Filters:

- LoggedChecked
- NotLoggedChecked
- NoCacher

Controllers (*servlets*):

- Login [not logged]
- Registration [not logged]
- GetAccountsData [logged]
- GetBankAccountDetailsData [logged]
- CreateBankAccount [logged]
- AddContacts [logged]
- MakeTransfer [logged]
- Logout [logged]
- GetContacts [logged]
- GetDestinations [logged]

Data access Objects (*DAO*):

- UserDAO
  - checkCredentials (email, pwd): User
  - getUserByID(id): User
  - getUserByEmail (email): User
  - getUserByUsername (username): User
  - registerUser (name, surname, email, username, pwd)
- BankAccountDAO
  - getBankAccountByID (id): BankAccount
  - findBankAccountsByUser (id_user): List<BankAccount>
  - findBankAccountByName (id_user, name_account): BankAccount
  - createBankAccount (id_user, name_account)
- MoneyTransferDAO
  - findMoneyTransferByAccountID (id_account): List<MoneyTransfer>
  - makeTransfer (id_bankAccount_src, id_user_dest, id_bankAccount_dest, reason, amount): int
  - getLastTransfer(): MoneyTransfer
  - getMoneyTransferByID(id): MoneyTransfer
- AddressBookDAO
  - getAddressBookByOwner(owner): AdressBook
  - addContact(owner, contactID)
  - existsContact(owner, contactID): boolean
  - getDestinations(owner): ArrayList<String>

Views (*templates*):

- index
- home

Client-side components

Index

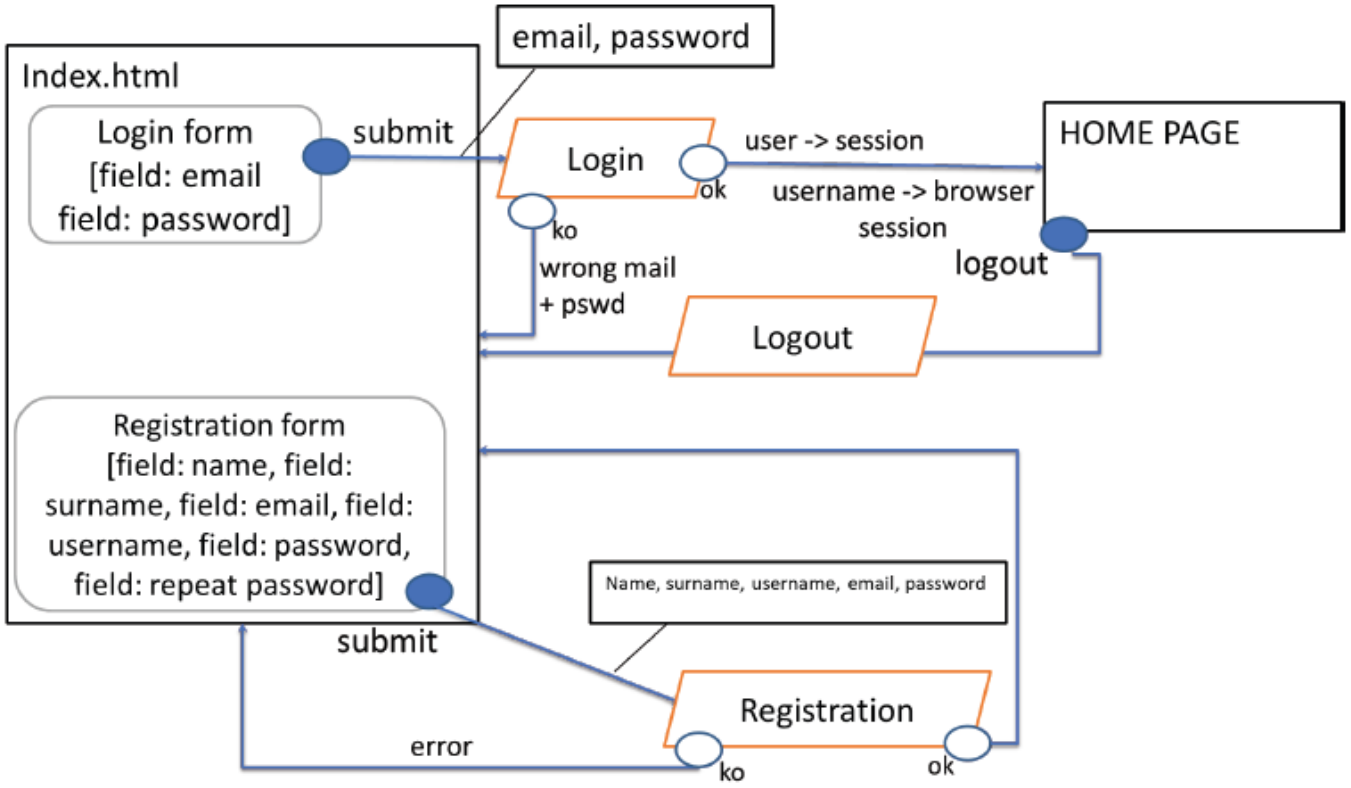
- Login form
  - Gestione del submit e degli errori
- Registration form
  - Gestione del submit e degli errori

Home

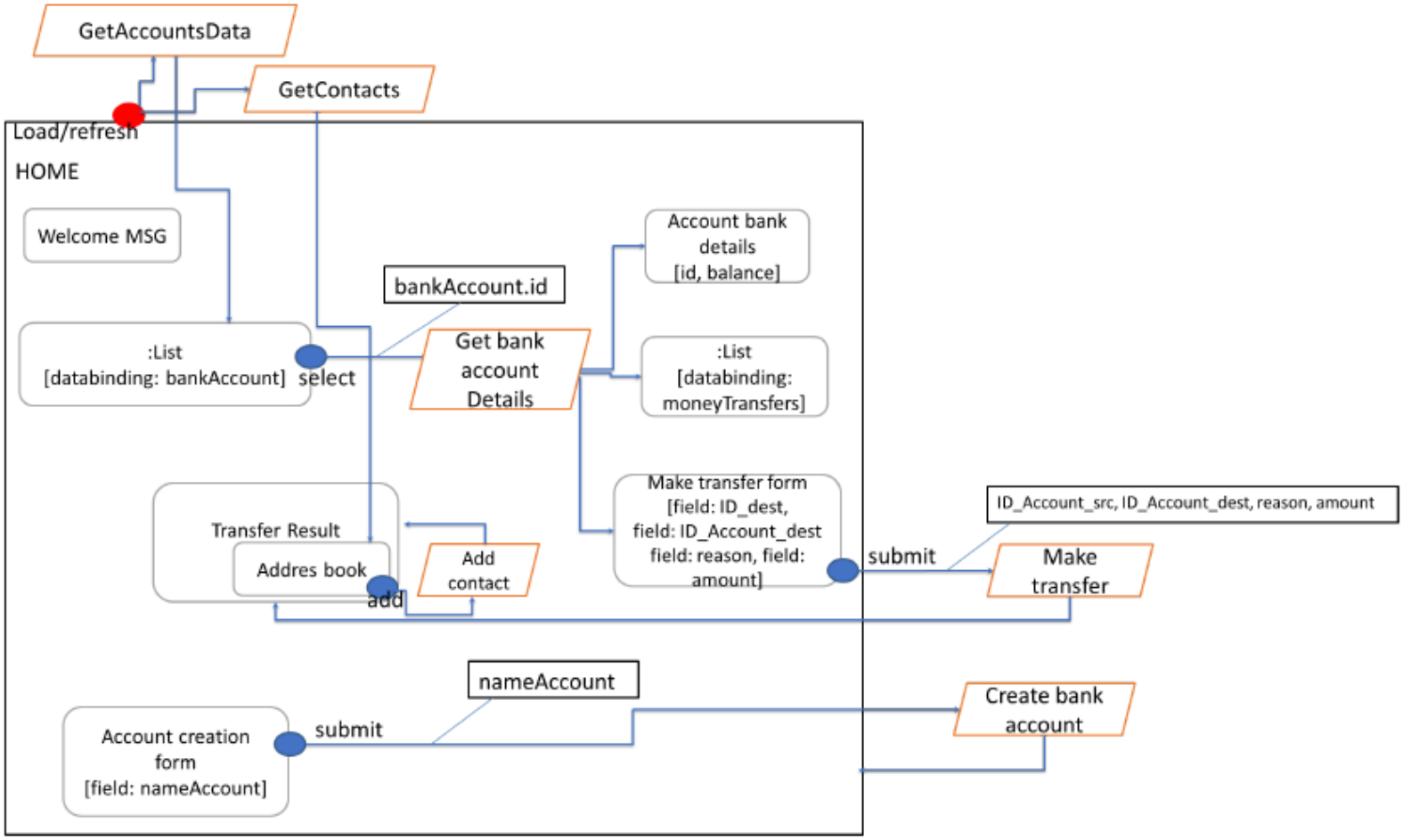
- PageOrchestrator
  - start(): inizializza i componenti della pagina
  - refresh(): carica i componenti della pagina
- PersonalMessage
  - show(): mostra il messaggio di benvenuto
- BankAccountList
  - show(): richiede al server i dati dell'elenco dei conti
  - update(): riceve i dati dal server e aggiorna la lista dei conti + registrazione di un handler per l'evento di click su ogni conto
  - autoclick(): seleziona un elemento della lista per mostrare in automatico i dettagli
  - reset(): nasconde la lista dei conti e la form per la creazione di un nuovo conto
- MoneyTransferList
  - show(id_account): carica i trasferimenti del conto selezionato
  - update(): riceve i dati dal server e aggiorna i dettagli del conto
  - reset(): nasconde i dettagli del conto e i trasferimenti, nasconde anche la form per fare i trasferimenti
- TransferResult
  - showSuccess(srcAccount, destAccount, transfer): mostra i dettagli del trasferimento e dei conti coinvolti
  - showFailure(reason): mostra il motivo del fallimento
  - reset(): nasconde i dettagli del successo/fallimento del trasferimento
- AddressBook
  - load(): carica i contatti dal server
  - showAddContactButton(srcUserID, destUserID): mostra il bottone per aggiungere un contatto se questo non è già in rubrica
  - addContact(srcUserID, destUserID): aggiunge un contatto in rubrica
  - autoCompleteNameDest(destUserName): mostra i suggerimenti per completare il campo di destinatario
  - autoCompleteAccountDest(): suggerisce gli account dell'utente selezionato per il trasferimento
  - reset(): nasconde il bottone per aggiungere un utente alla rubrica
  - getDestinationUsers(): carica i nomi degli utenti in rubrica

Design applicativo (IFML)

Login – Registration



Home


GetAccountsData
GetContacts
Load/refresh
HOME
Welcome MSG
bankAccount.id
Account bank details [id, balance]
:List [databinding: bankAccount]
select
Get bank account Details
:List [databinding: moneyTransfers]
Transfer Result
Addres book
add
Add contact
Make transfer form [field: ID_dest, field: ID_Account_dest field: reason, field: amount]
submit
ID_Account_src, ID_Account_dest, reason, amount
Make transfer
nameAccount
Account creation form [field: nameAccount]
submit
Create bank account

Eventi & Azioni

N.B. i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
index → login form → submit	Controllo dati (validità mail)	POST mail, password	Controllo credenziali
index → registration form → submit	Controllo dati (validità mail + uguaglianza password)	POST name, surname, username, mail, pwd	Registrazione
Home page→ load	Aggiorna view con elenco conti e con i contatti dell'utente	GET	Estrazione conti e rubrica utente
Home → bank accounts → select account	Aggiorna view con dettagli sul conto ed i trasferimenti	GET (id conto)	Estrazione dettagli conto e trasferimenti
Home → transfer form → submit	Controllo dati (importo>0, importo<balance)	POST (dati trasferimento)	Controlla ed emetti risposta
Home → success transfer	Mostra i dettagli del trasferimento	-	-
Home → success transfer → add contact	AJAX POST	POST (info contatto)	Aggiungi contatto
Home → transfer form → insert input	Mostra suggerimenti	-	-
Home → create account form → submit	Controllo nome	POST (nome conto)	Controllo e creazione conto
Logout	Cancellazione della sessione dal browser	GET	Terminazione della sessione

Controller – Event handler

N.B. makeCall indica una funzione che fa una chiamata asincrona al server

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
index → login form → submit	Function makeCall	POST mail, password	Login
index → registration form → submit	Function makeCall	POST name, surname, username, mail, pwd	Registration
Home page→ load	Function PageOrchestrator →BankAccountList →AddressBook	 GET GET	 GetBankAccountsData GetContacts
Home → bank accounts → select account	Function BankAccountList →MoneyTransferList	 GET (account id)	 GetBankAccountDetailsData
Home → transfer form → submit	Function MoneyTransferList	POST (dati trasferimento)	MakeTransfer
Home → success transfer	Function TransferResult	-	-
Home → success transfer → add contact	Function AddressBook	POST (info contatto)	AddContact
Home → transfer form → typing on destCode	Function AddressBook	-	-
Home → transfer form → typing on destAccount	Function AddressBook	-	-
Home → create account form → submit	Function BankAccountList	POST (nome conto)	CreateBankAccount
Logout	Function logout	GET	Logout

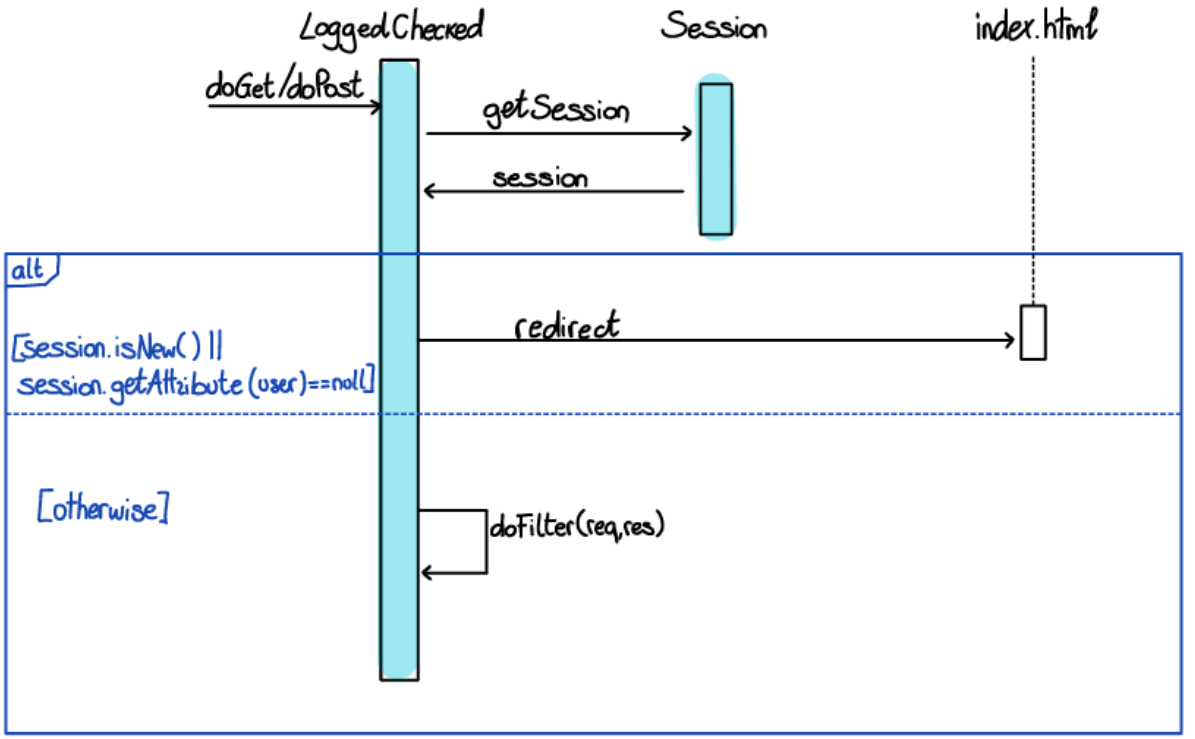
Sequence diagrams

Some controls over parameters and potential internal errors and database access have been omitted for abbreviation.

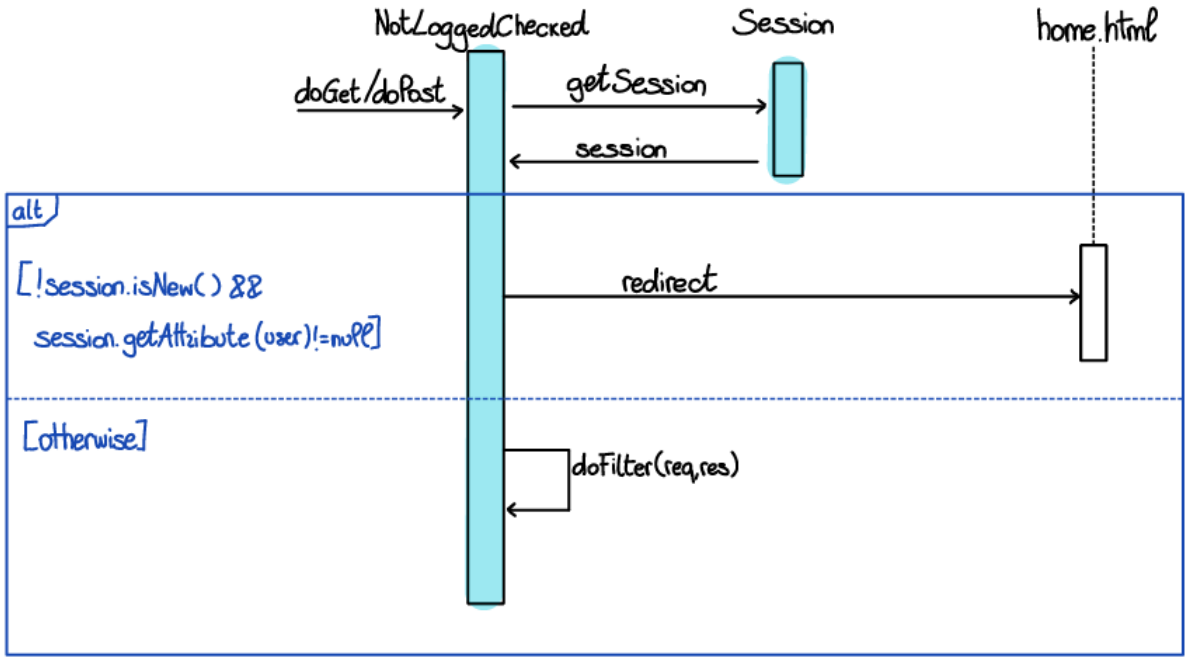
The controls conducted by filters have been presented in the first diagrams and omitted in the following ones.

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

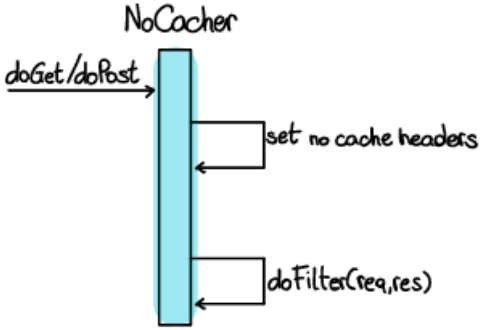
Logged Checked



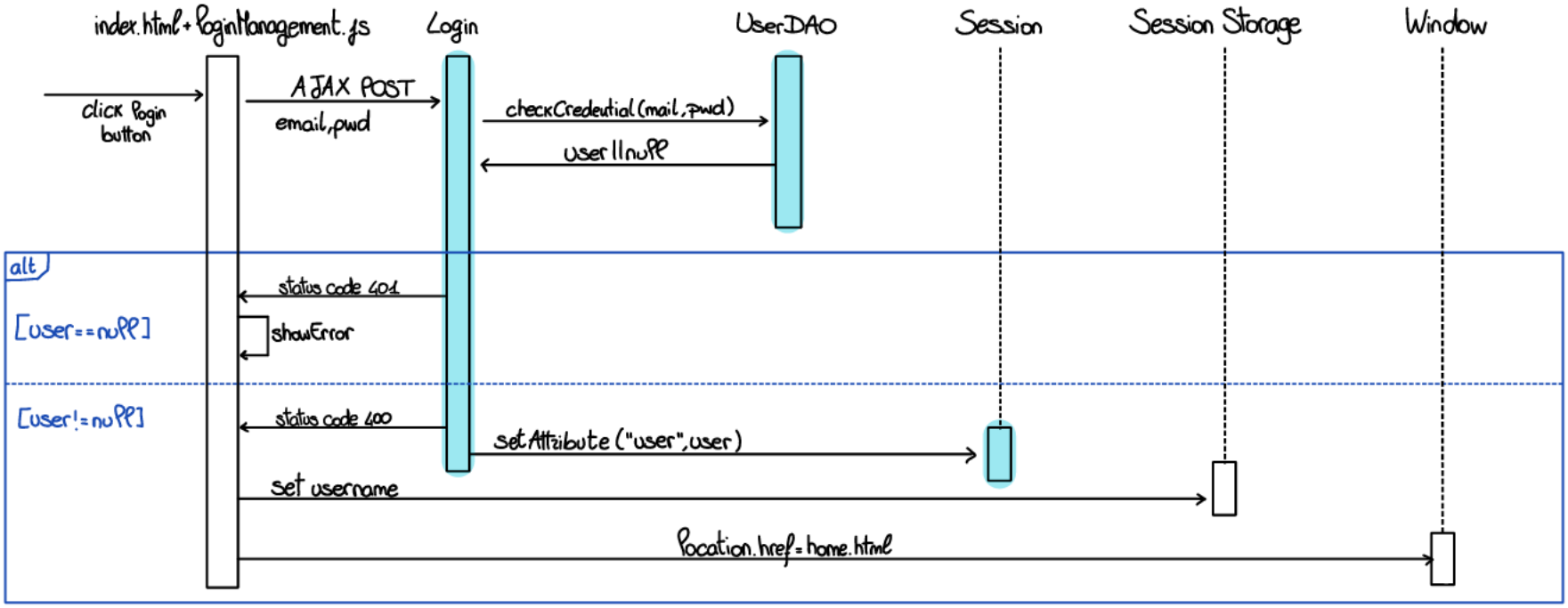
Not Logged Checked



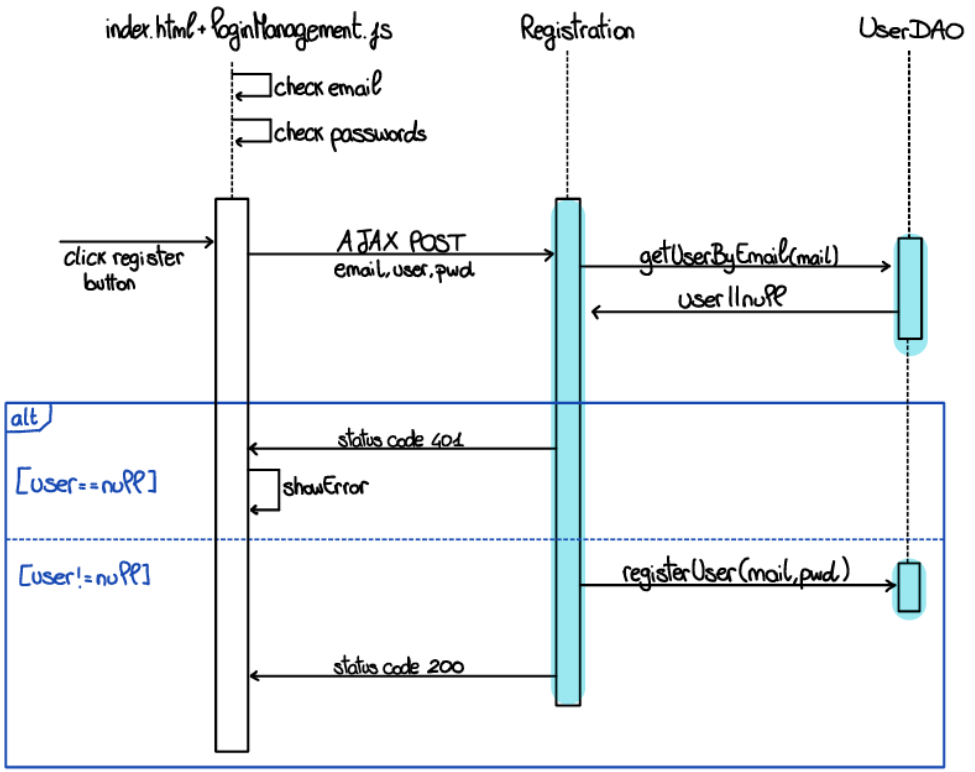
NoCacher



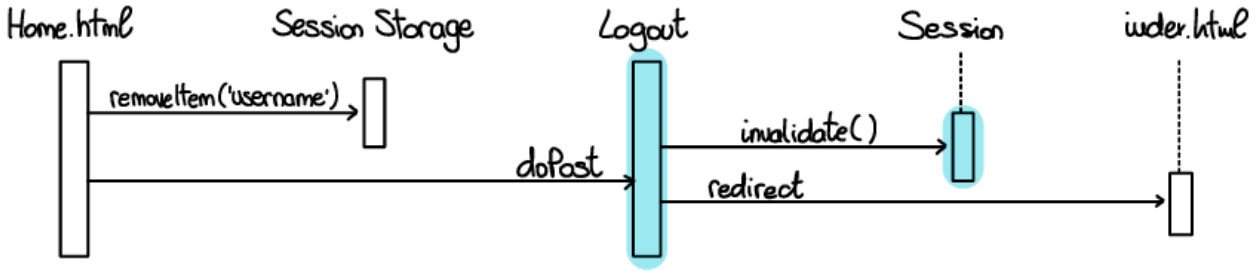
Evento: Login



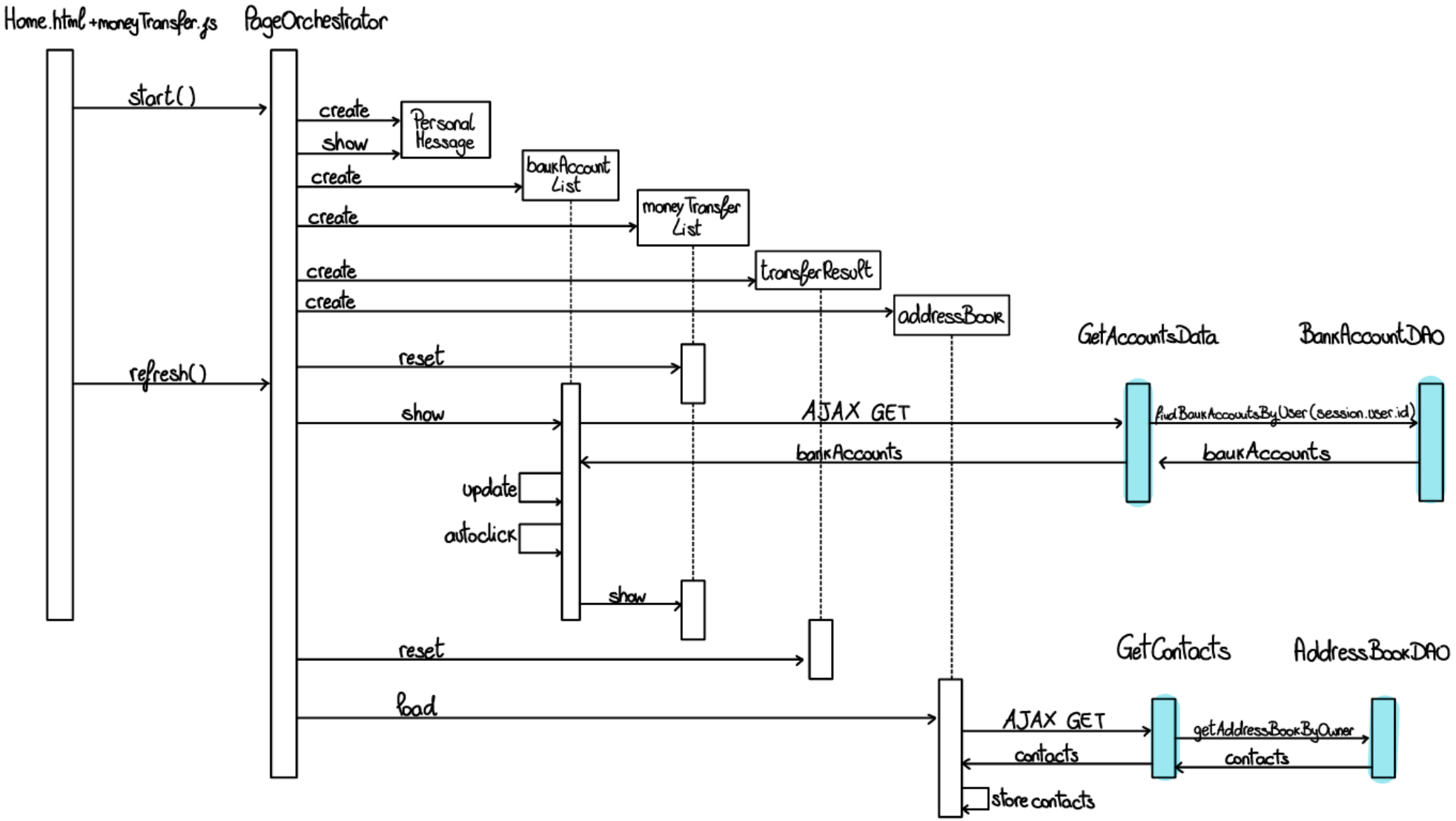
Evento: Registrazione



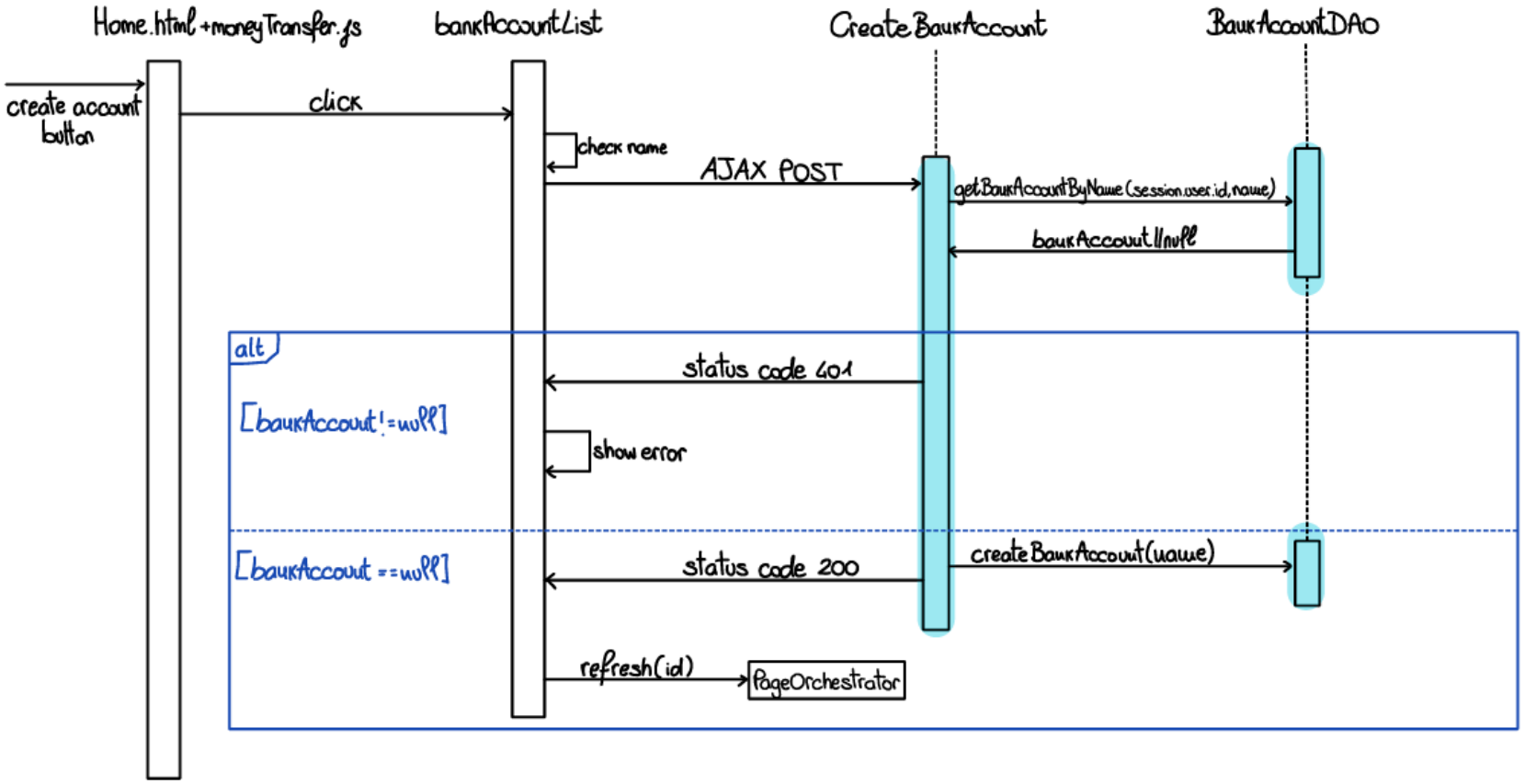
Evento: Logout



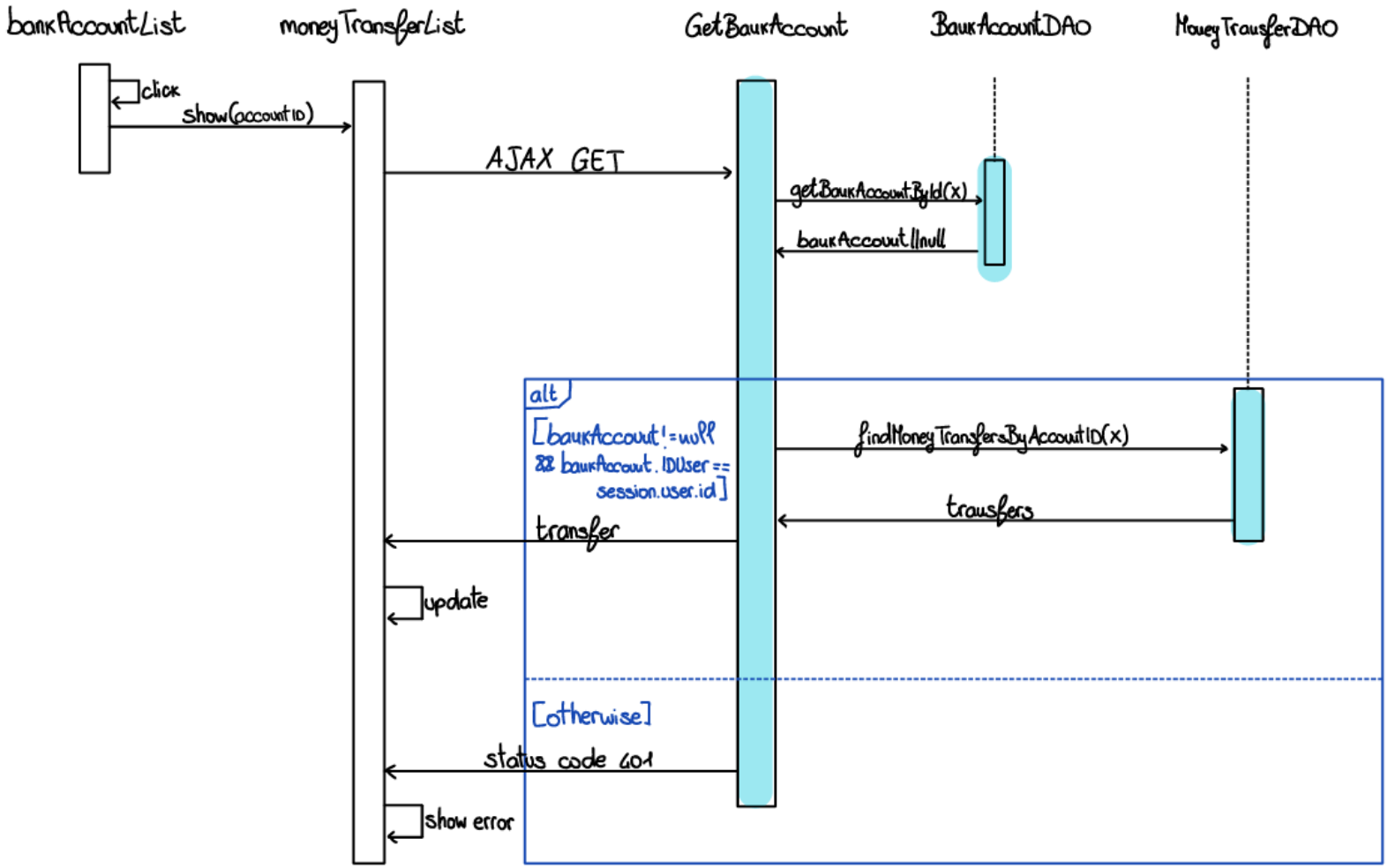
Evento: caricamento home page



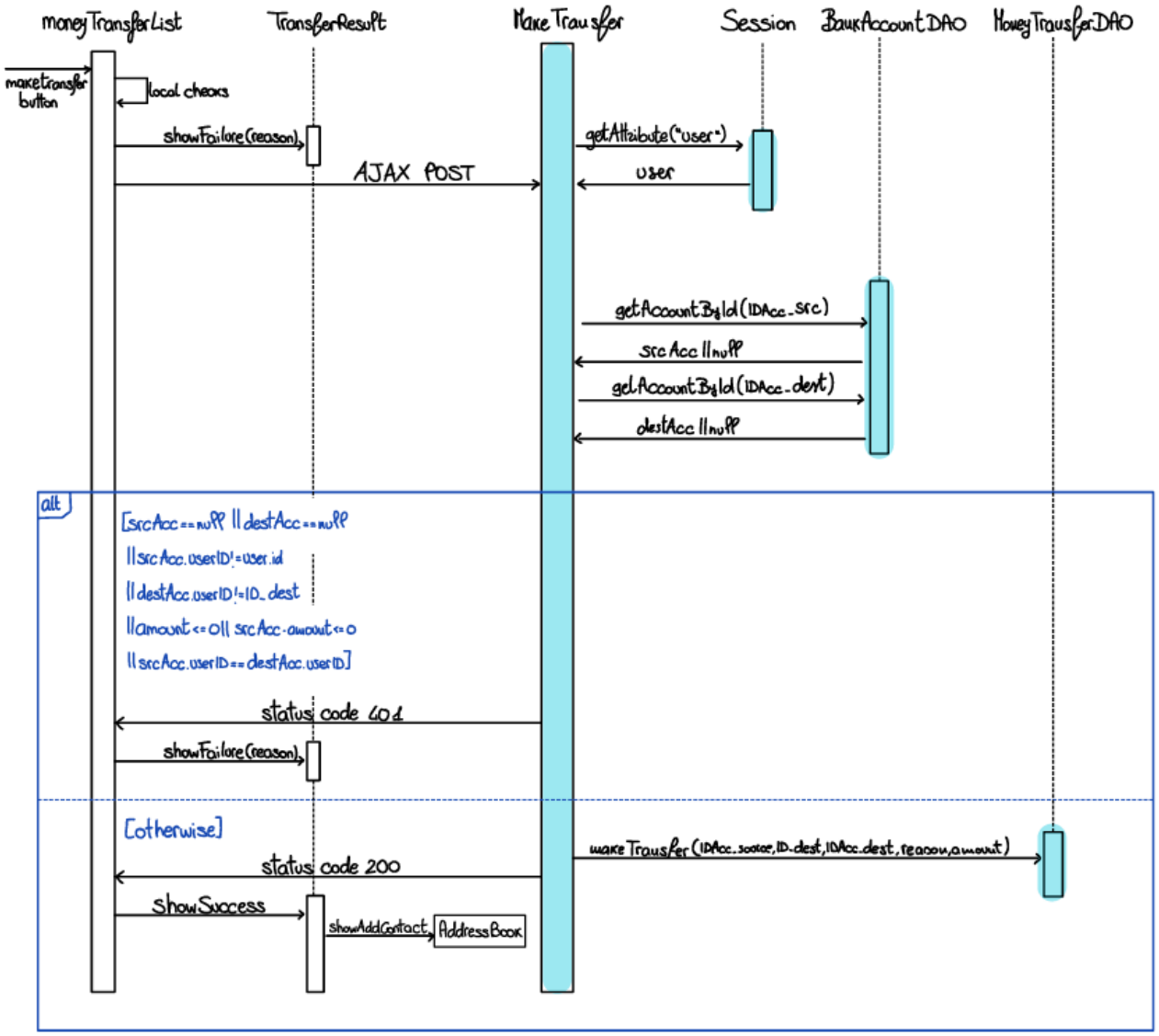
Evento: creazione conto



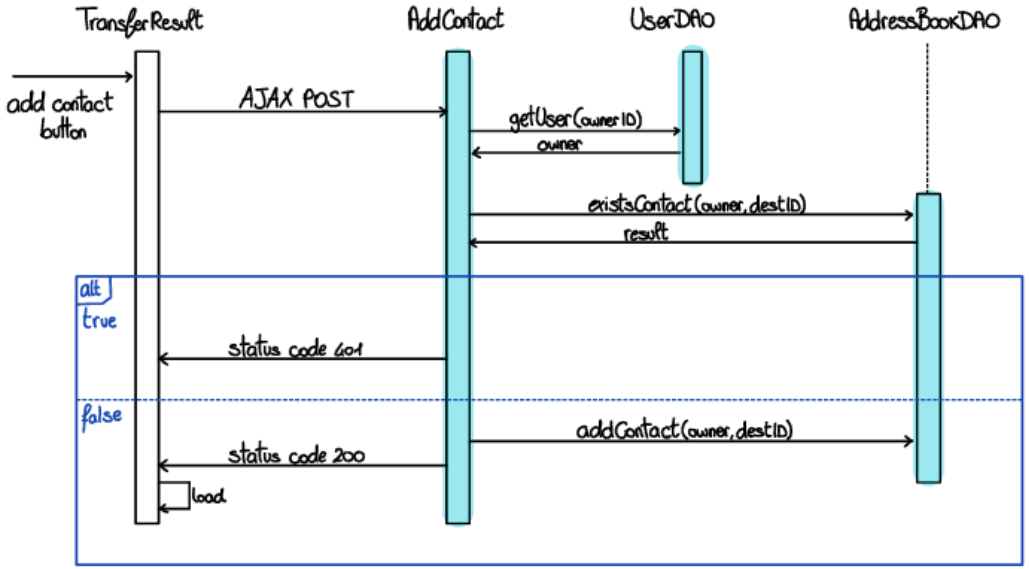
Evento: selezione di un conto



Evento: trasferimento



Evento: aggiungi contatto



Evento: auto completamento