Esercizio 4 - RIA

Giovanni Manfredi – Sebastiano Meneghin

Esercizio 4: trasferimento denaro (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.

Esercizio 4: trasferimento denaro (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.

Analisi dati per database (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.

Entità, attributi, relazioni

Analisi dati per database (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.

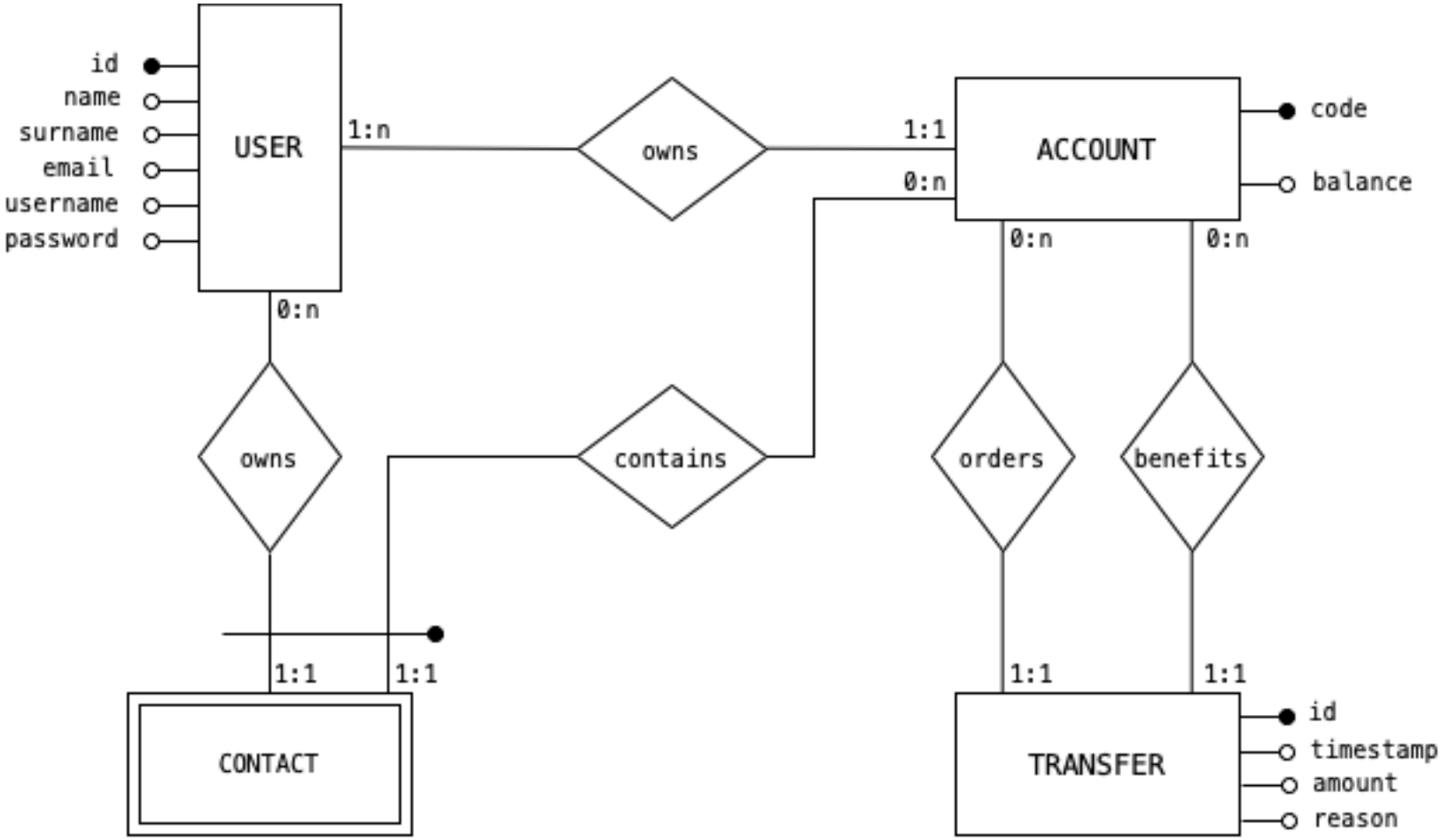
Entità, attributi, relazioni

Completamento delle specifiche (1/3) - database

- Il campo *email* è necessariamente univoco. Utenti diversi non possono avere la stessa email
- Il campo *username* e *email* sono distinti e entrambi univoci. Vanno specificati dall'utente nella registrazione
- Il saldo (*balance*) di ogni conto (*account*) deve essere in qualsiasi momento maggiore o uguale a zero
- L'importo (*amount*) di un trasferimento (*transfer*) deve essere sempre maggiore di zero
- Il conto di origine (*code_account_orderer*) di un trasferimento (*transfer*) non può corrispondere al conto di destinazione (*code_account_beneficiary*) del trasferimento
- La data (*timestamp*) del trasferimento non può essere anteriore al momento in cui questo viene ordinato
- Per creare la rubrica si è deciso lato DB di salvare un'entità debole detta *contact* che crea una corrispondenza tra un proprietario del contatto (*owner_id*) e l'account (*contact_account*) di chi è stato contattato

Entità, attributi, relazioni

Database design



Database schema (1/3)

```
CREATE TABLE `user` (
  `id` int unsigned NOT NULL AUTO_INCREMENT,
  `name`     varchar(45) NOT NULL,
  `surname`  varchar(45) NOT NULL,
  `email`    varchar(320) NOT NULL,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `user_username_un` UNIQUE
(`username`),
  CONSTRAINT `user_email_un` UNIQUE (`email`)
) ENGINE=InnoDB
```

```
CREATE TABLE `account` (
  `code` int unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int unsigned NOT NULL,
  `balance` decimal(15,2) unsigned NOT NULL
DEFAULT '0.00',
  PRIMARY KEY (`code`),
  CONSTRAINT `account_user_id_fk`
   FOREIGN KEY (`user_id`)
   REFERENCES `user` (`id`)
    ON UPDATE CASCADE
    ON DELETE NO ACTION,
  CONSTRAINT `account_balance_ck`
   CHECK (`balance` >= 0)
) ENGINE=InnoDB
```

Database schema (2/3)

```
CREATE TABLE `transfer` (
  `id`                      int unsigned NOT NULL AUTO_INCREMENT,
  `timestamp`               timestamp NOT NULL DEFAULT current_timestamp,
  `account_code_orderer`    int unsigned NOT NULL,
  `account_code_beneficiary` int unsigned NOT NULL,
  `amount` decimal(15,2)    unsigned NOT NULL,
  `reason` varchar(255)     NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `transfer_account_code_beneficiary_fk` FOREIGN KEY (`account_code_beneficiary`)
   REFERENCES `account` (`code`) ON UPDATE CASCADE ON DELETE NO ACTION,
  CONSTRAINT `transfer_account_code_orderer_fk` FOREIGN KEY (`account_code_orderer`)
   REFERENCES `account` (`code`) ON UPDATE CASCADE ON DELETE NO ACTION,
  CONSTRAINT `transfer_amount_ck` CHECK (`amount` > 0)
) ENGINE=InnoDB
```

Database schema (3/3)

```
CREATE TABLE `contact` (
  `owner_id`        int unsigned NOT NULL,
  `contact_account` int unsigned NOT NULL,
  PRIMARY KEY (`owner_id`,`contact_account`),
  CONSTRAINT `contact_owner_id_fk` FOREIGN KEY (`owner_id`)
   REFERENCES `user` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `contact_contact_account_fk` FOREIGN KEY (`contact_account`)
   REFERENCES `account` (`code`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB
```

Database content (1/2)

user

id	name	surname	email	username	password
1	Giovanni	Manfredi	giovanni@polimi.it	Gio	passwordDifficile23
2	Sebastiano	Meneghin	sebastiano@polimi.it	Seba	passwordDifficile25
3	Jeff	Bezos	boss@amazon.com	Jeffrey	youCanDoIt
4	Jim	Gray	admin@ibm.com	Admin	admin

account

code	user_id	balance
1	1	1000.05
2	2	2000.99
3	3	1000000.85
4	3	2000000.95
5	1	285.60
6	4	4000

contact

owner_id	contact_account
4	1
3	1
1	4

Database content (2/2)

transfer

id	timestamp	code_account_orderer	code_account_beneficiary	amount	reason
1	2022-03-03 00:05:55	3	1	1000.00	project funding
2	2022-04-05 10:35:01	5	2	500.00	cash
3	2022-08-05 03:35:01	3	6	200.00	refund

Analisi requisiti applicazione (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.

Pagine, componenti, eventi, azioni

Analisi requisiti applicazione (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.

Pagine, componenti, eventi, azioni

Completamento delle specifiche (2/3) - applicazione

- Le uniche due pagine presenti sono LOGIN e HOME
- La pagina di LOGIN contiene il form per la registrazione a scomparsa tramite appositi bottoni (*hide and show*)
- Nella HOME ad ogni conto (account) in elenco viene affiancato il saldo (balance) del conto
- Per credenziali di login si intendono email e password
- Se un utente prova ad accedere alla pagina di LOGIN ed è ancora loggato, verrà reindirizzato automaticamente alla HOME
- Nella pagina HOME è anche possibile per l'utente creare un nuovo conto (tramite un apposita sezione *hide and show*)
- Se un nuovo utente si registra verrà creato automaticamente un conto a lui collegato con saldo a zero. Non è quindi possibile che un utente non abbia un account a lui collegato
- Dopo la registrazione l'utente dovrà comunque fare login tramite l'apposito form (compare un messaggio di operazione avvenuta dopo essersi registrati)

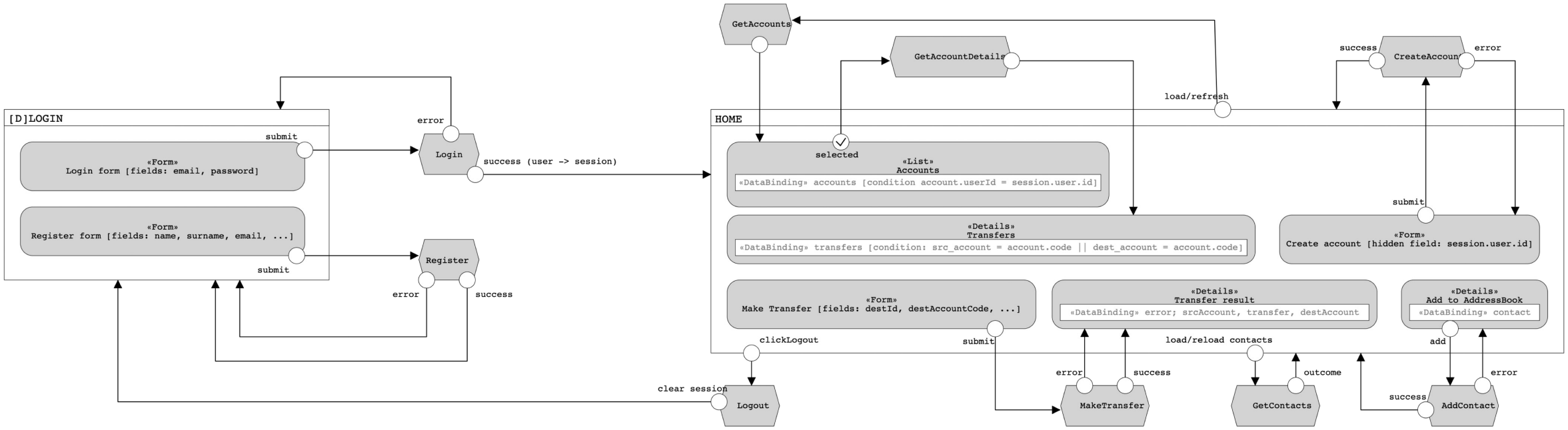
Pagine, componenti, eventi, azioni

Completamento delle specifiche (3/3) - applicazione

- 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
- Per migliorare l'esperienza utente, per ogni form viene eseguita la cattura del tasto *ENTER* che esegue un invio del form come se fosse stato premuto il pulsante a schermo *SUBMIT*
- Ad ogni richiesta AJAX (asincrona) al server, apparirà un messaggio di risposta nella parte in basso a sinistra dello schermo
- Diversi elementi nella HOME, vista la necessità di avere tutto su una sola pagina, possono essere nascosti o mostrati tramite appositi tasti
- Per semplificare l'accesso ai contatti (*contact*) dell'utente è stato deciso di utilizzare una struttura più complessa a livello di beans e DAO detta *AddressBook*, composta dal campo *owner_id* e da una mappa che mette in corrispondenza i proprietari dei conti con i loro conti (*Map <user_id, account_code>*).

Pagine, componenti, eventi, azioni

Design applicativo (IFML)



Riferirsi ai sequence diagrams per lo sviluppo dettagliato delle interazioni nell'applicazione

Server-side components

- Model objects (beans)
  - User
  - Account
  - Transfer
  - AddressBook
- Data Access Objects (classes)
  - UserDAO
    - findUser(email, password) : User
    - findUserById(id) : User
    - findUserByEmail(email) : User
    - findUserByUsername(username) : User
    - createUser(name, surname, email, username, password) : void
    - registerUser (name, surname, email, username, password) : void
    - registerUser (name, surname, email, username, password, balance) : void
  - AccountDAO
    - findAccountByCode(code) : Account
    - findAccountsByUserId(user_id) : List<Account>
    - createAccount(user_id) : void
    - createAccount(user_id, balance) : void
  - TransferDAO
    - findTransfersByAccountCode(code) : List<Transfer>
    - createTransfer(code_account_orderer, code_account_beneficiary, amount, reason) : void
- Filters
  - CheckLoggedUser
  - CheckNotLoggedUser
  - NoCacher
  - AddressBookDAO
    - findAddressBookByOwnerId(int ownerId) : AddressBook
    - doesContactExits(int ownerId, int contactAccount) : boolean
    - createContact(int ownerId, int contactAccount): void
- Controllers (serlvets)
  - Login
  - Logout
  - Register
  - GetAccounts
  - GetContacts
  - AddContact
  - CreateAccount
  - GetAccountDetails
  - MakeTransfer
- Views (templates)
  - login.html
  - home.html

Client-side components

- Login (Index)
  - Login form
    - manage submit and errors
  - Register form
    - manage submit and errors
- Home
  - PageOrchestrator
    - start(): initializes page components
    - refresh(excludeContacts): loads page components, excluding loading the address book if the flag is set
  - UserInfo
    - show(): shows user's data in page header
  - AccountList
    - show(): loads accounts
    - update(): refreshes accounts' view with accounts' data
  - TransferList
    - show(accountCode): loads transfers of the provided account code
    - hide(): hides account details view
    - update(): refreshes account details view with transfers' data
  - TransferResult
    - showSuccess(srcAccount, transfer, destAccount): shows transfer's outcome in case of success
    - showFailure(reason): shows transfer's failure reason
  - AddressBook
    - load(): loads contacts from server
    - showButton(destUserCode, destAccountCode): shows add contact button when contact is not already in the address book
    - addContact(destUserCode, destAccountCode): add contact to address book
    - autocompleteDest(destUserCode): provides autosuggestion options for destination users for transfers
    - autocompleteAccount(destUserCode, destAccountCode, currentAccount): provides autosuggestion options for destination accounts for transfers (excluding current account)

Eventi e Azioni

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
Login.html → login form → submit	Data check	POST (username, password)	Data check
Login.html → register form → submit	Password match and data check	POST (credentials)	Data check
Home.html → load	Update view → update account and contacts	2 GET no param	Retrieval of data from db and Json conversion
Home.html → account list → account selection	Update view and display transfers of selected account	GET (account id)	Retrieval of data from db and Json conversion
Press enter in an input field	Click the correspondent custom submit button	-	-
From hide/show buttons	Hide/show correspondent form	-	-
Home.html → Make transfer → submit	Data check	POST (transfer data)	Transfer check and outcome
Home.html → successResult → addContact	Ajax post	POST (contact info)	Addition of contact
Home.html → Transfer list → transfer success	Optionally show «add contact» button & show success	-	-
Home.html → Account list → create account	-	POST ()	Data check & insertion
Home.html → Transfer list → insert input in make transfer form	Process and display suggestions	-	-

Eventi e Controllers

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
Index → login form → submit	Function makeCall	POST (username, password)	Login (servlet)
Index → register form → submit	Function makeCall	POST (name, surname, email, username, password)	Register (servlet)
Home → load	Function PageOrchestrator → Account List → AddressBook	 GET () GET ()	 GetAccounts (servlet) GetContacts (servlet)
Home → account list → select account	Function AccountList → TransferList	 GET (accountCode)	 GetAccountDetails (servlet)
Home → account form → submit	Function AccountList	POST ()	CreateAccount (servlet)
Home → trasfer form → submit	Function TransferList	POST (transfer data)	MakeTransfer (servlet)
Home → transfer success div → add contact	Function AddressBook	POST (contact data)	AddContact (servlet)
Home → transfer form → click/typing on destId	Function TransferList → AddressBook	-	-
Home → transfer form → click/typing on destAccountCode	Function TransferList → AddressBook	-	-

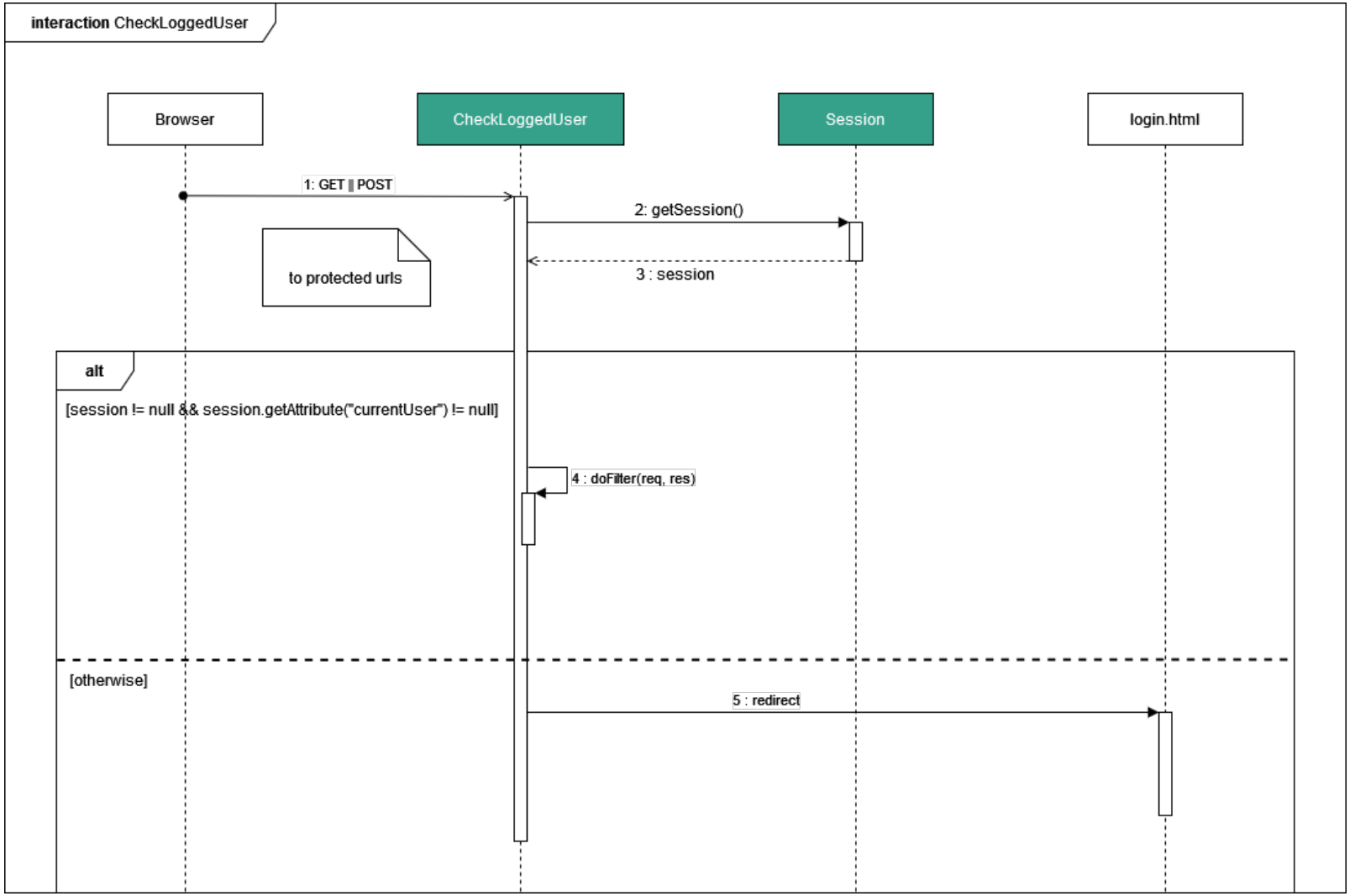
Sequence diagrams

I diagrammi di sequenza seguenti mirano a illustrare l'interazione degli eventi principali dell'applicazione web. Si è cercato di utilizzare il maggior grado di dettaglio possibile nella rappresentazione ma, a favore di una migliore chiarezza e minore ripetitività, alcuni elementi sono stati omessi dopo la loro prima rappresentazione (es. errori interni al server, errori di accesso interni al database, parametri *null*).

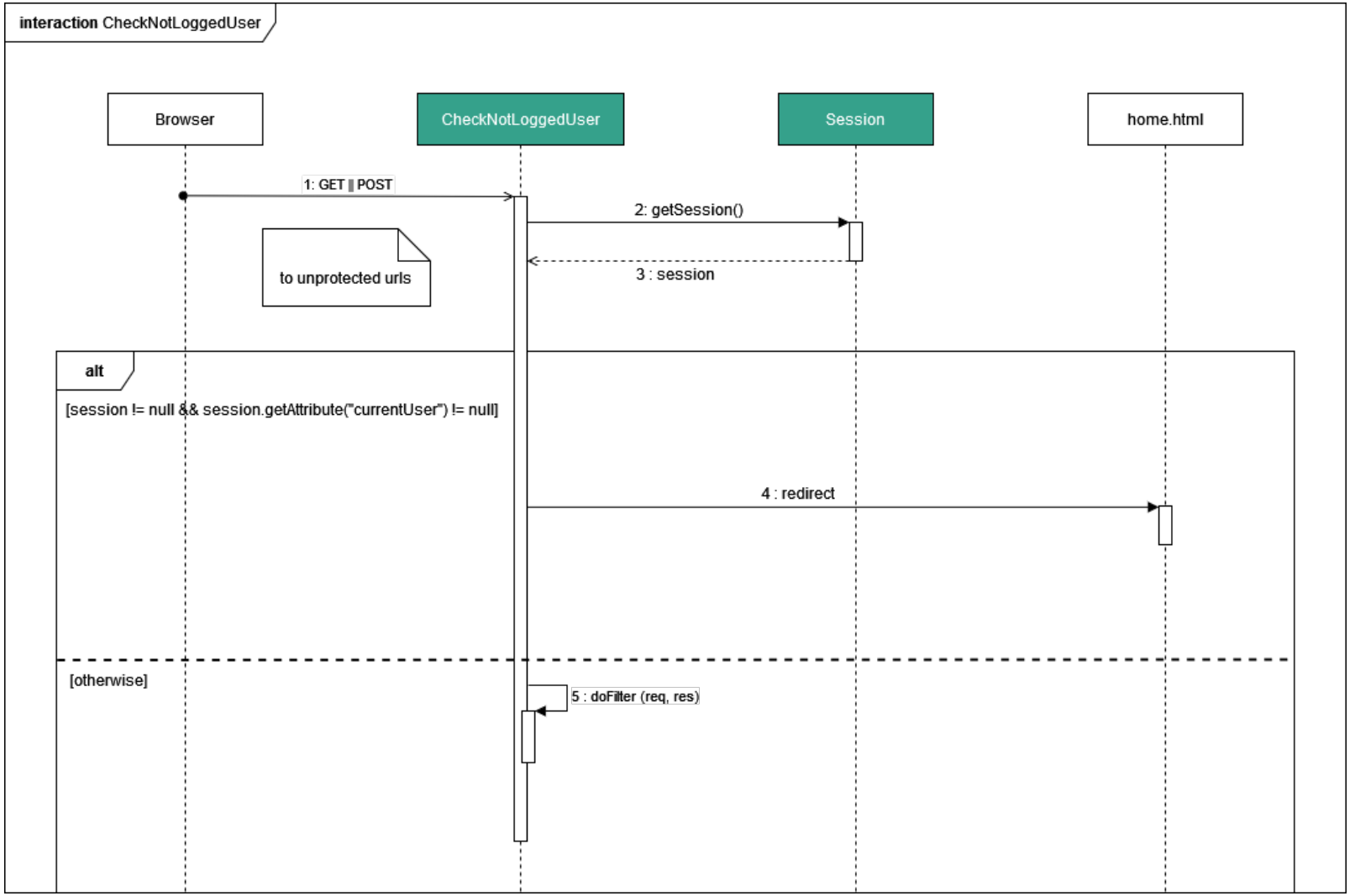
Inoltre i controlli effettuati sui filtri rappresentati nei primi diagrammi sono stati omessi nelle chiamate successive, che vengono però svolte correttamente. Per un maggior dettaglio sulle chiamate dei filtri si consiglia di visionare il file `web.xml`.

Per facilitare la distinzione tra server e client i riquadri degli *objects* del server sono stati evidenziati con questo colore.

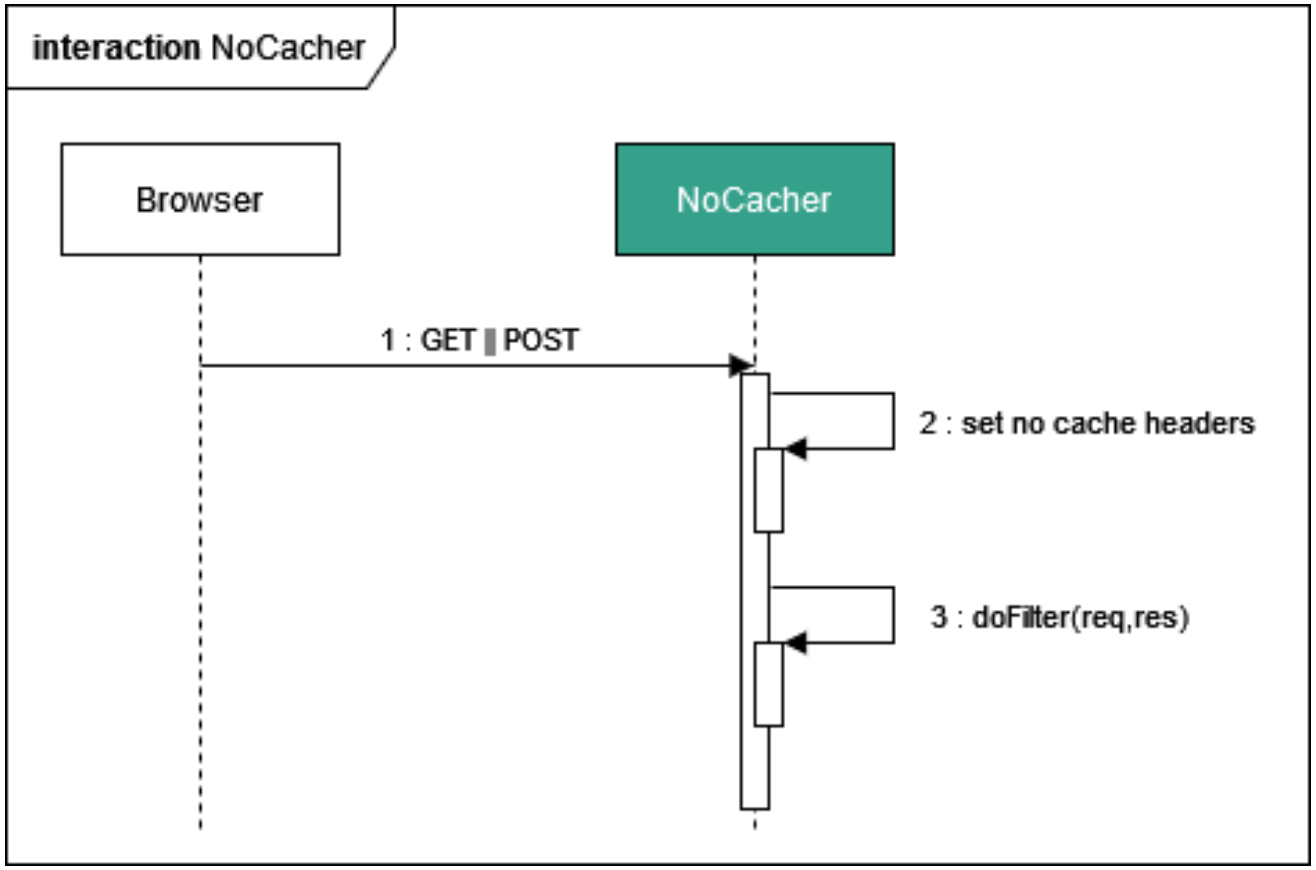
CheckLoggedUser



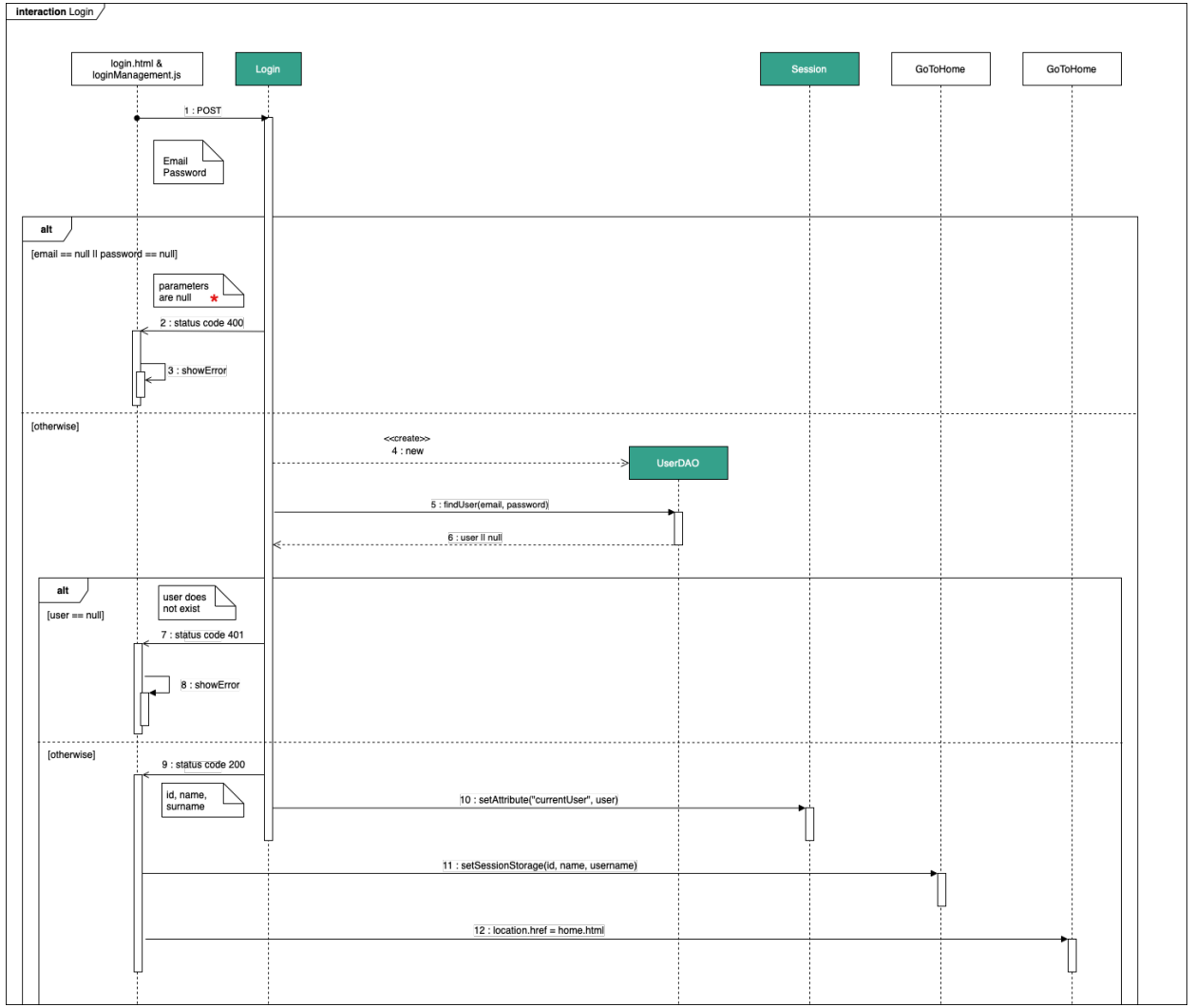
CheckNotLoggedUser



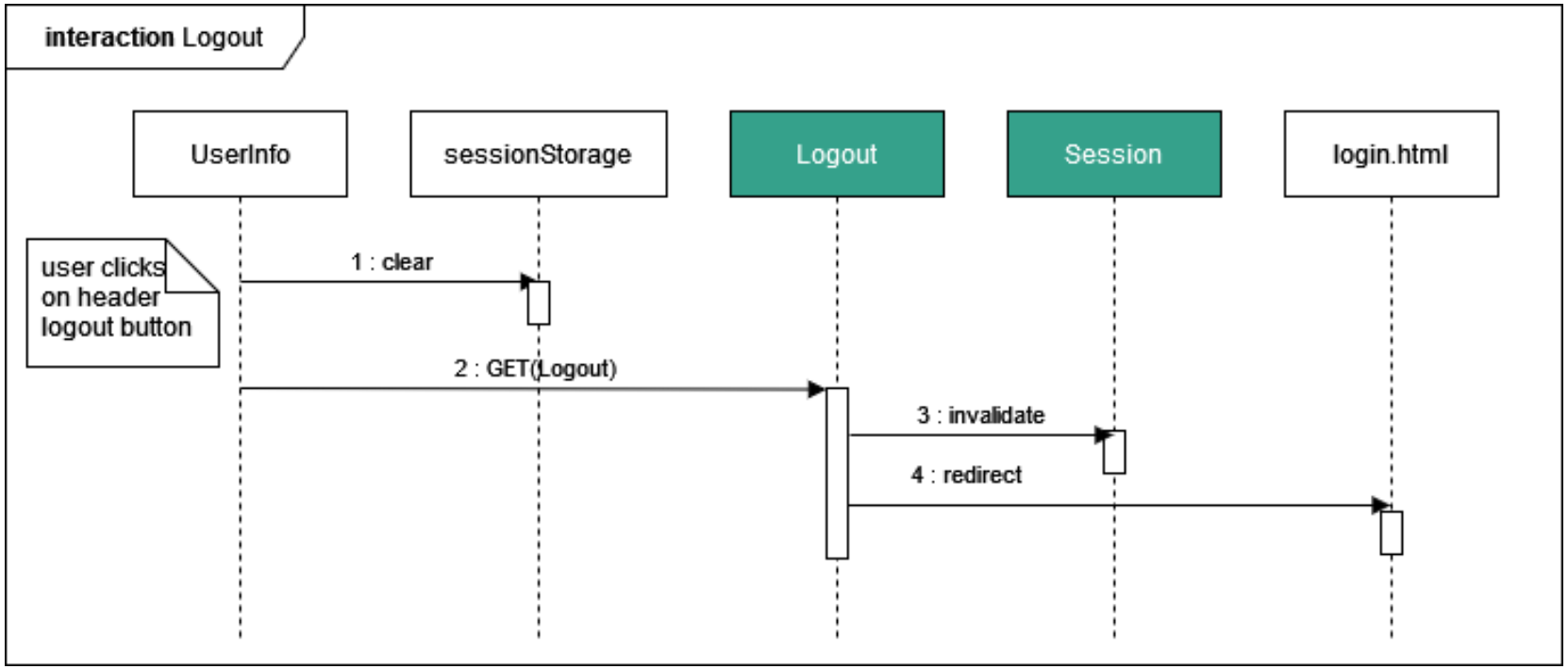
NoCacher


interaction NoCacher
Browser
NoCacher
1 : GET | POST
2 : set no cache headers
3 : doFilter(req,res)

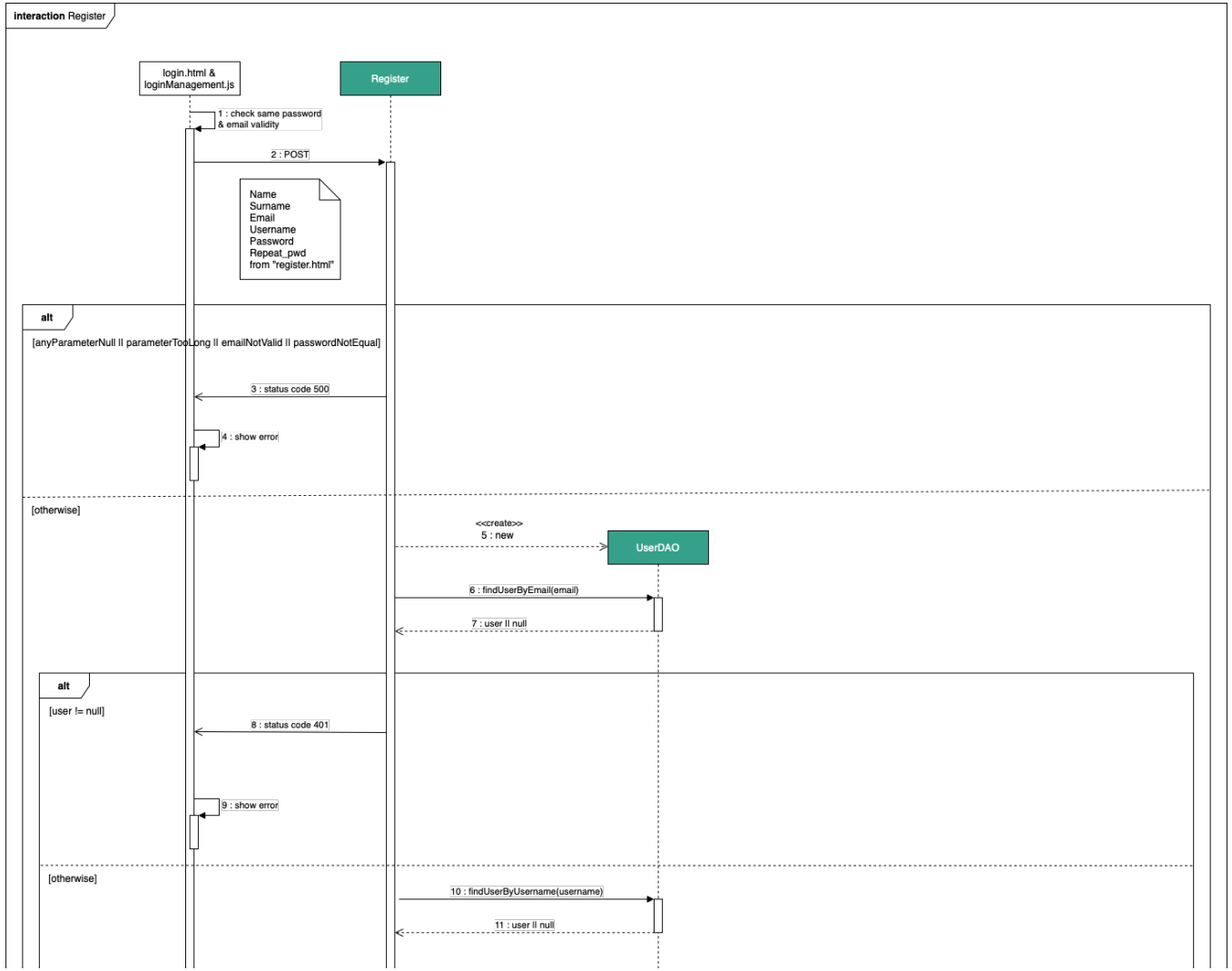
Login


interaction Login
login.html & loginManagement.js
Login
Session
GoToHome
GoToHome
1 : POST
Email
Password
alt
[email == null || password == null]
parameters are null
2 : status code 400
3 : showError
[otherwise]
<<create>>
4 : new
UserDAO
5 : findUser(email, password)
6 : user || null
alt
[user == null]
user does not exist
7 : status code 401
8 : showError
[otherwise]
9 : status code 200
id, name, surname
10 : setAttribute("currentUser", user)
11 : setSessionStorage(id, name, username)
12 : location.href = home.html

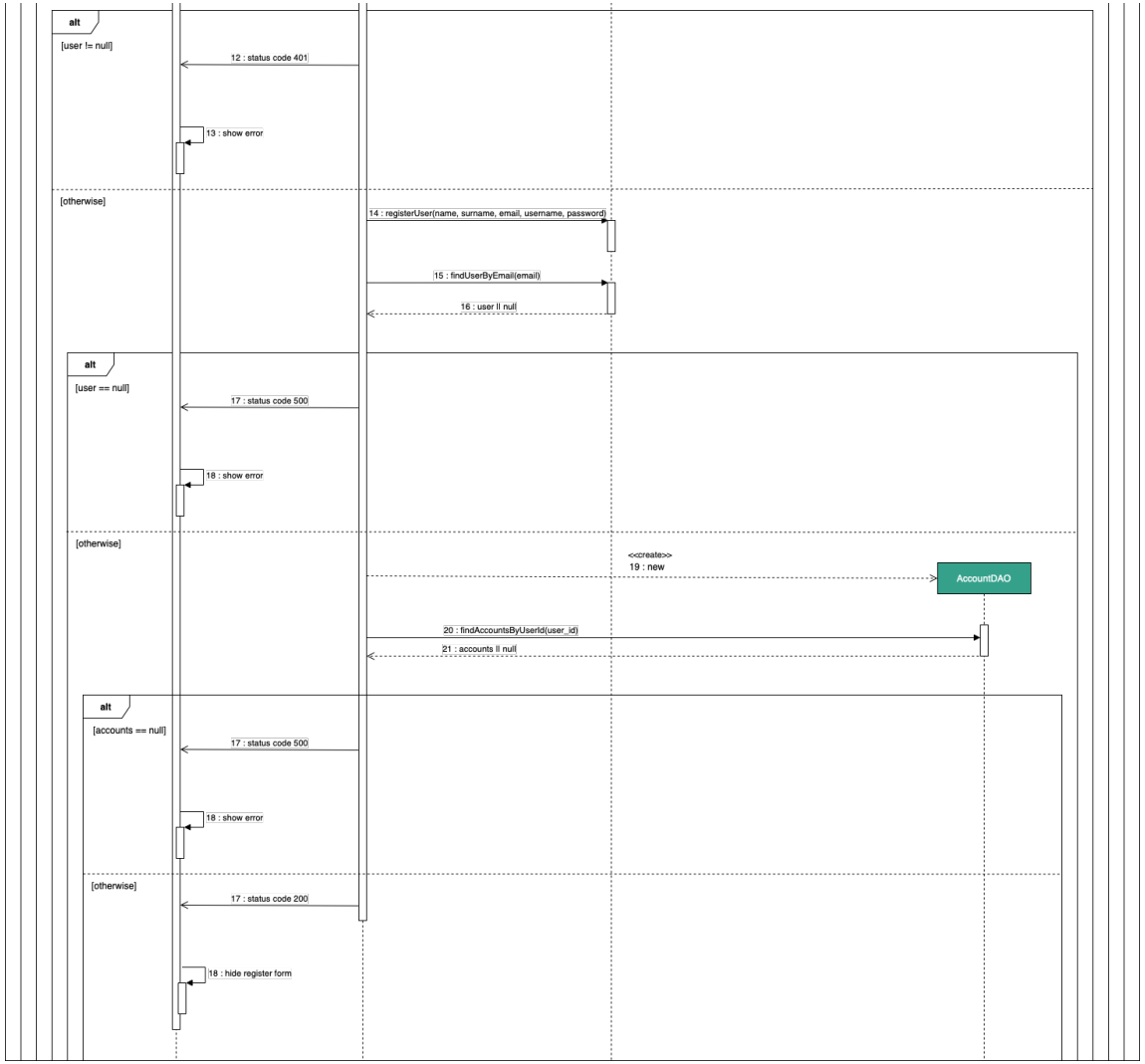
Logout


interaction Logout
UserInfo
sessionStorage
Logout
Session
login.html
user clicks on header logout button
1 : clear
2 : GET(Logout)
3 : invalidate
4 : redirect

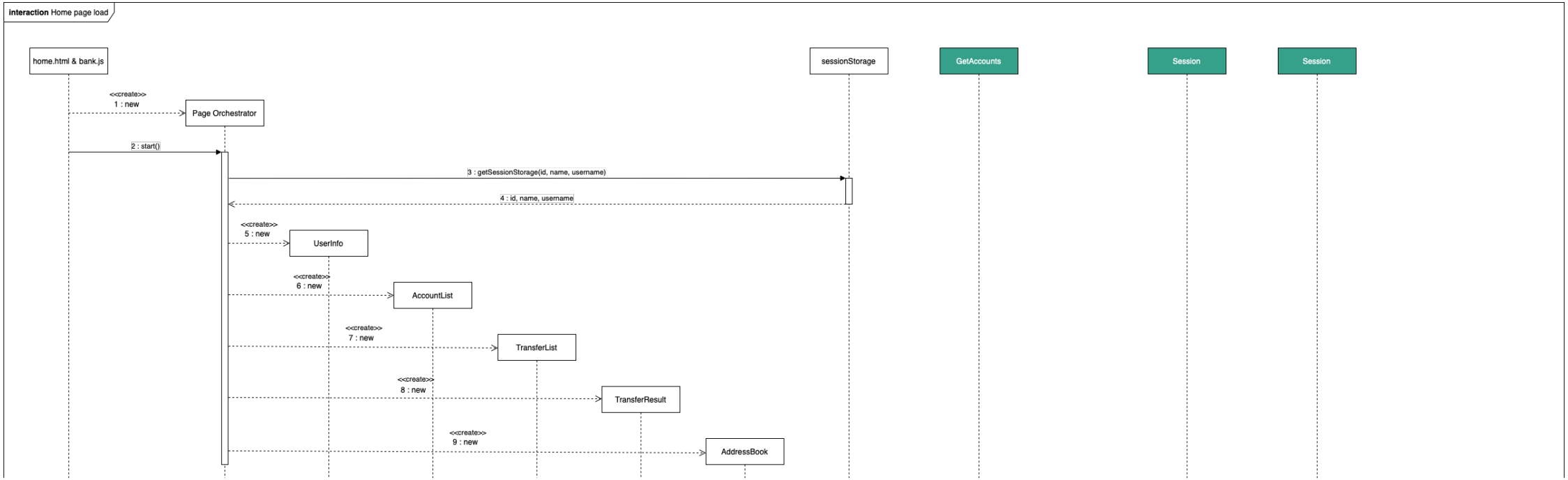
Register (1/2)


interaction Register
login.html & loginManagement.js
Register
1 : check same password & email validity
2 : POST
Name
Surname
Email
Username
Password
Repeat_pwd
from "register.html"
alt
[anyParameterNull || parameterTooLong || emailNotValid || passwordNotEqual]
3 : status code 500
4 : show error
[otherwise]
<<create>>
5 : new
UserDAO
6 : findUserByEmail(email)
7 : user || null
alt
[user != null]
8 : status code 401
9 : show error
[otherwise]
10 : findUserByUsername(username)
11 : user || null

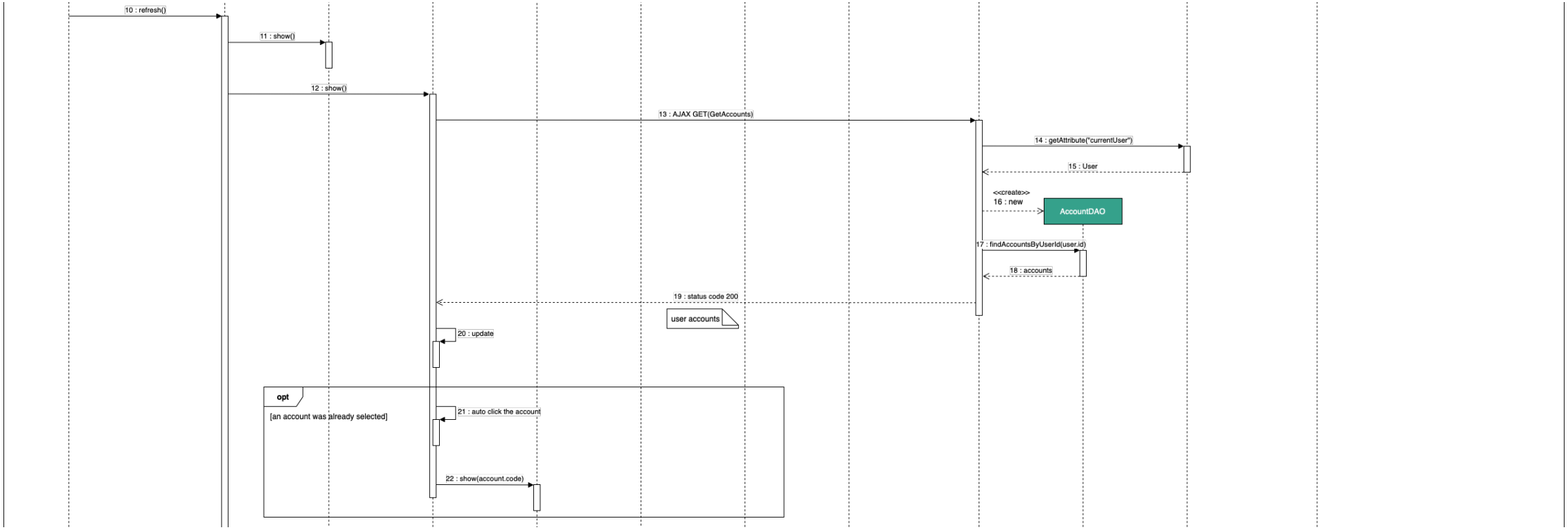
Register (2/2)


alt
[user != null]
12 : status code 401
13 : show error
[otherwise]
14 : registerUser(name, surname, email, username, password)
15 : findUserByEmail(email)
16 : user || null
alt
[user == null]
17 : status code 500
18 : show error
[otherwise]
<<create>>
19 : new
AccountDAO
20 : findAccountsByUserId(user_id)
21 : accounts || null
alt
[accounts == null]
17 : status code 500
18 : show error
[otherwise]
17 : status code 200
18 : hide register form

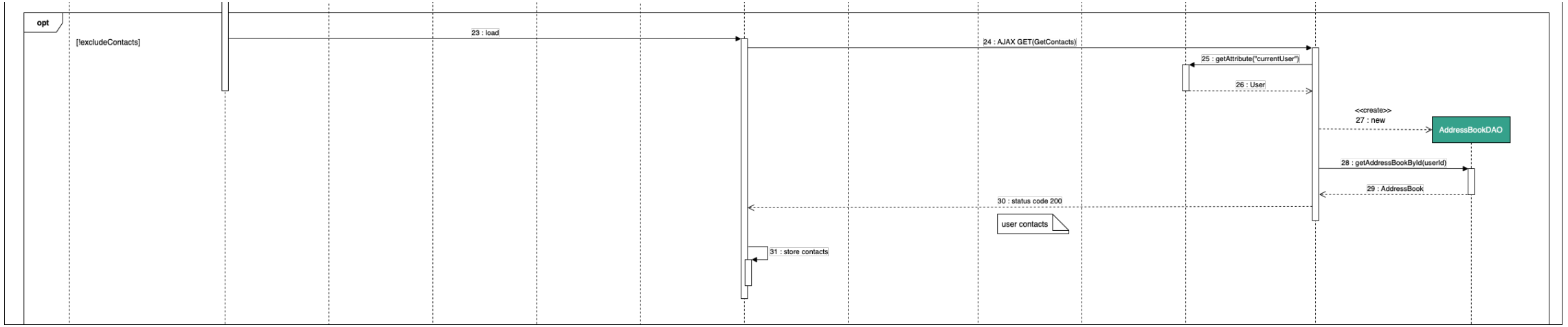
Home page load (1/3)



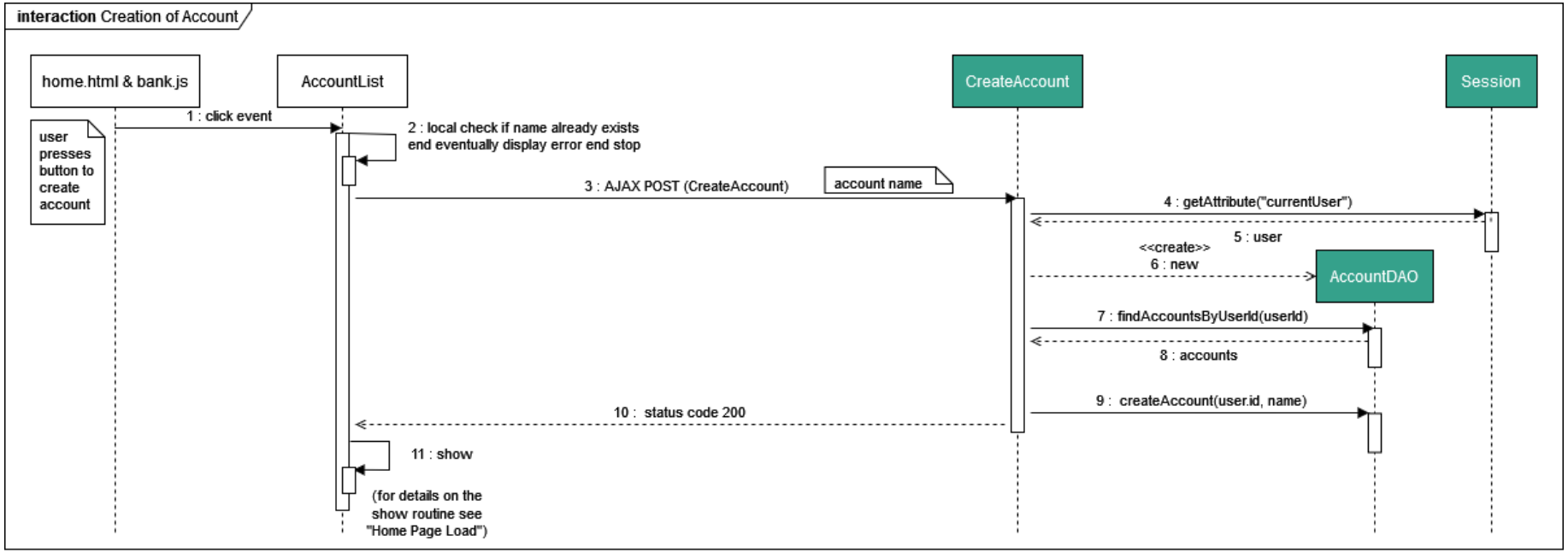
Home page load (2/3)



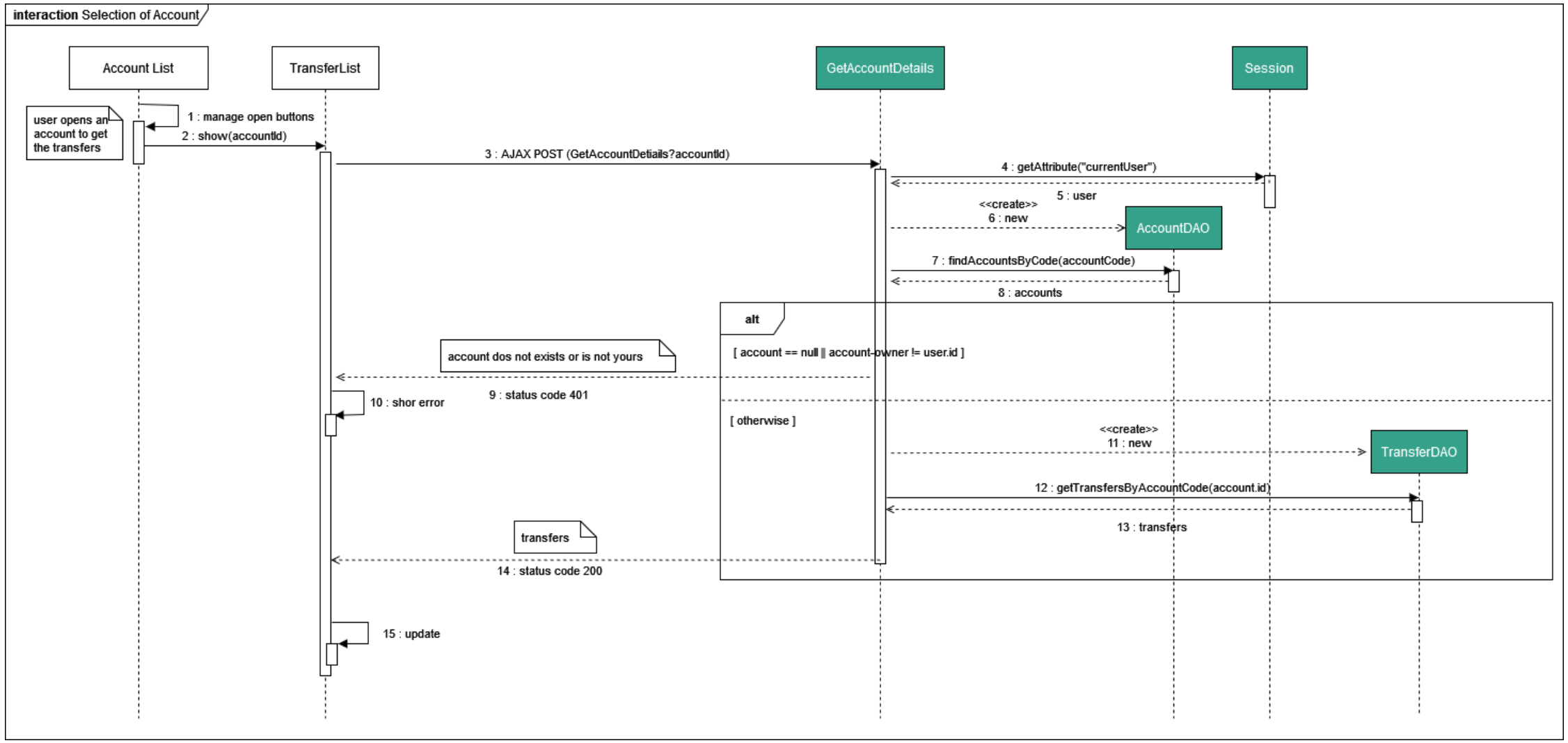
Home page load (2/3)



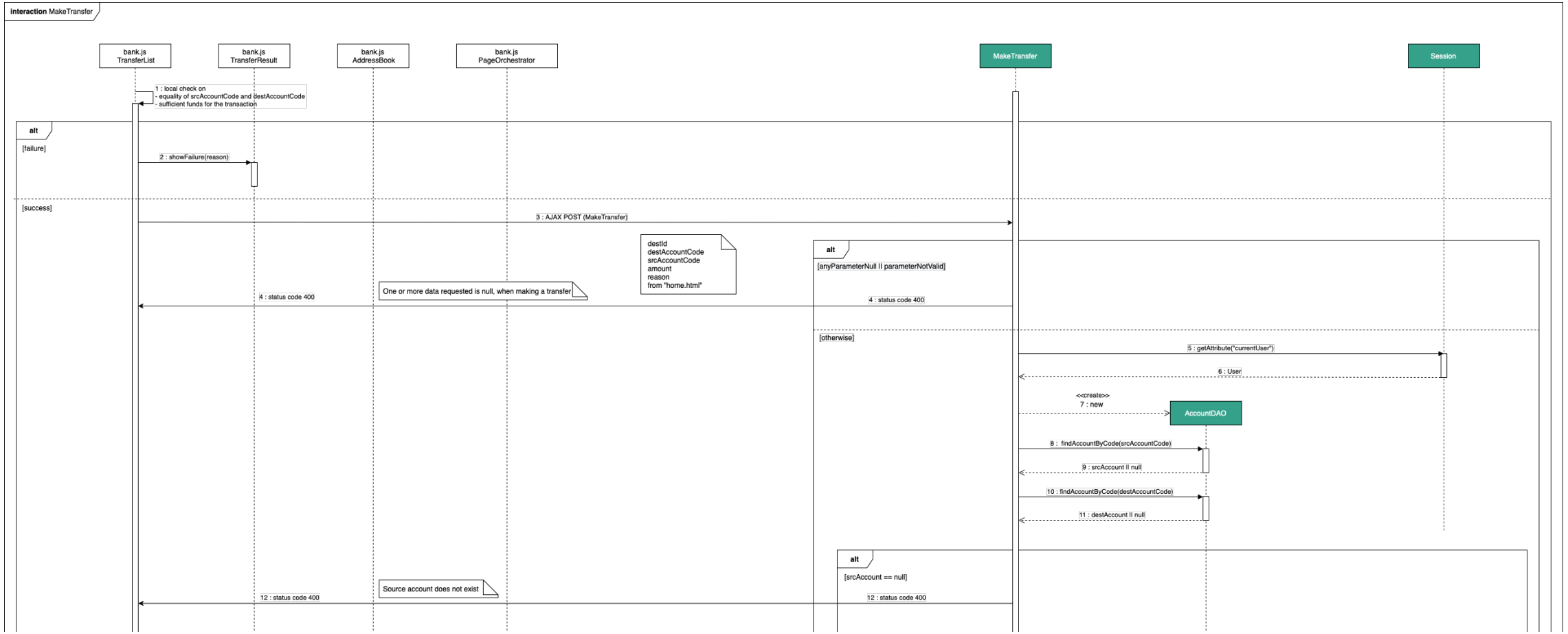
CreateAccount



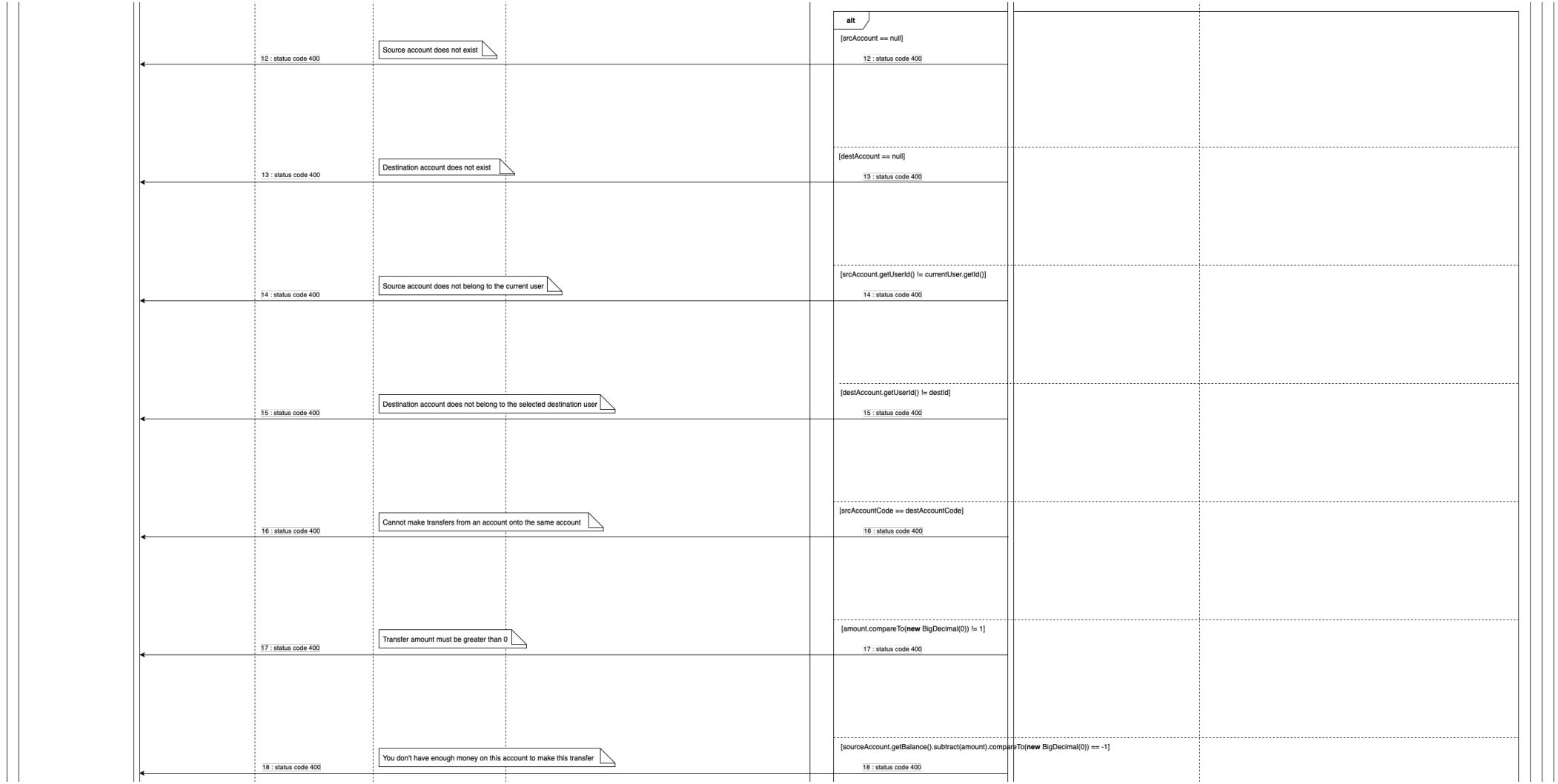
Selection of an account



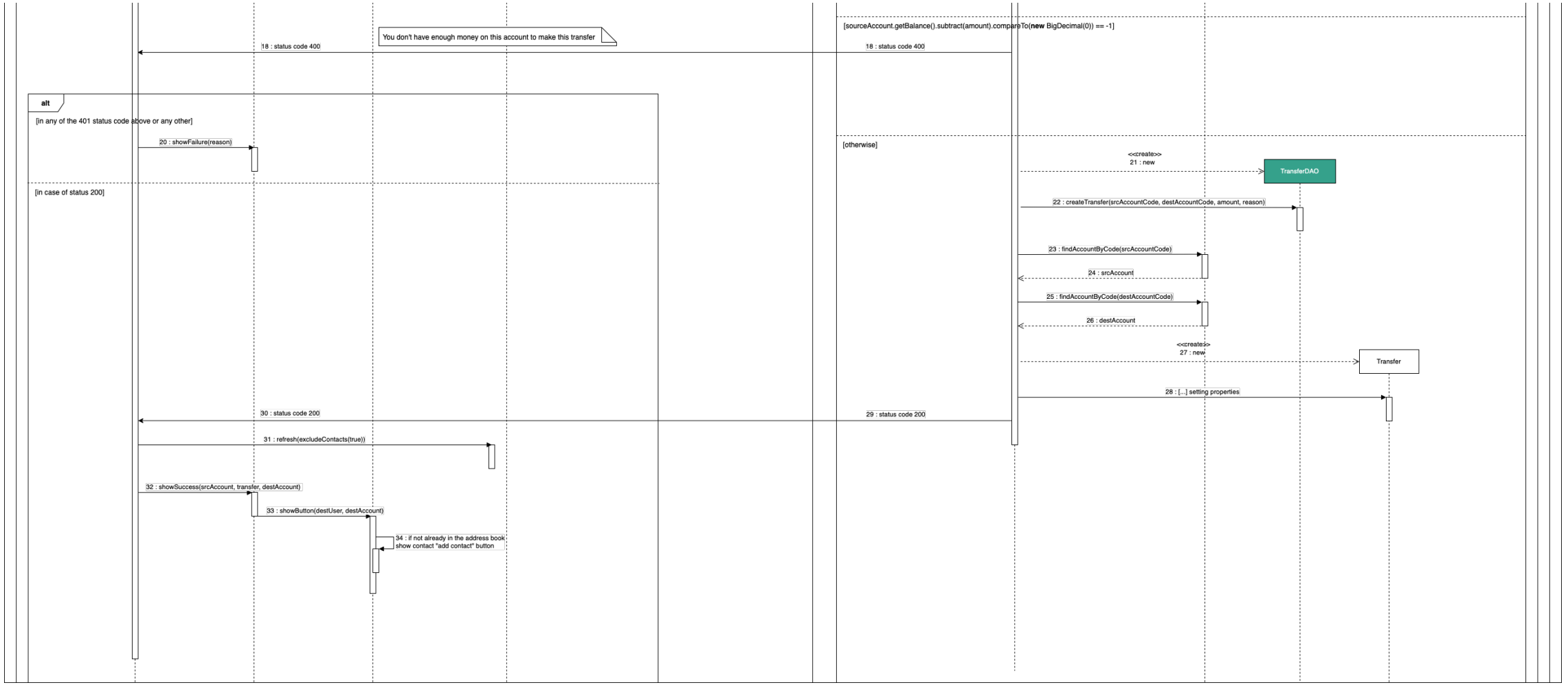
MakeTransfer (1/3)



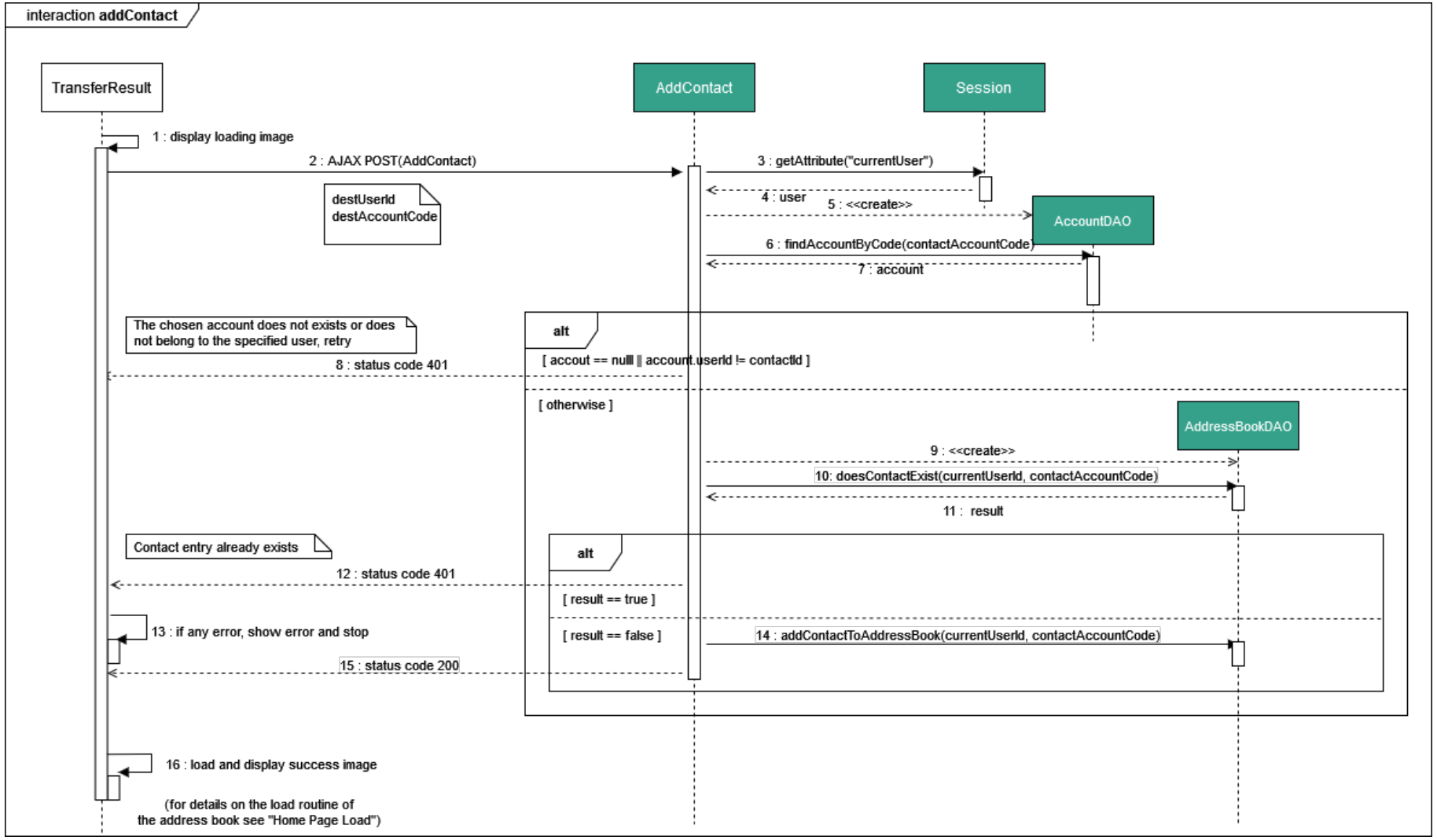
MakeTransfer (2/3)



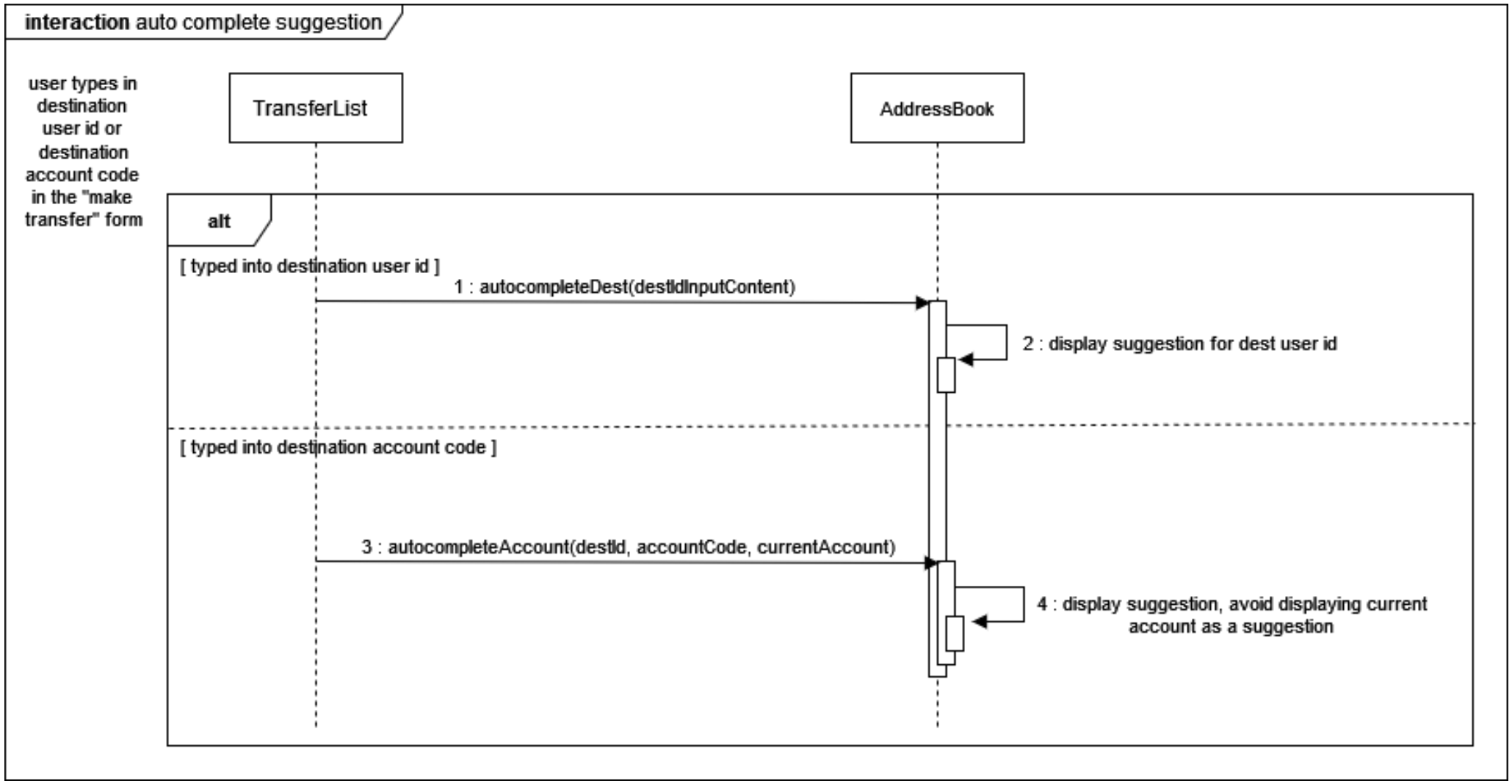
MakeTransfer (3/3)



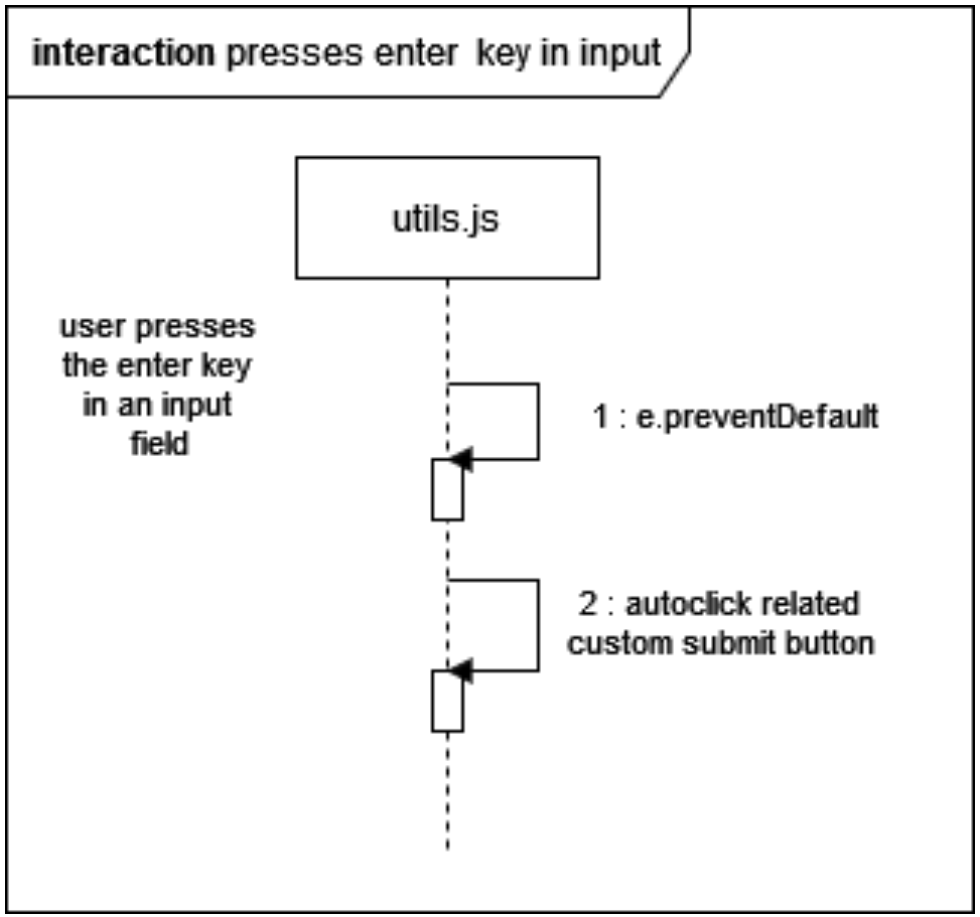
AddContact


interaction addContact
TransferResult
AddContact
Session
AccountDAO
AddressBookDAO
1 : display loading image
2 : AJAX POST(AddContact)
destUserId
destAccountCode
3 : getAttribute("currentUser")
4 : user
5 : <<create>>
6 : findAccountByCode(contactAccountCode)
7 : account
The chosen account does not exists or does not belong to the specified user, retry
8 : status code 401
alt
[accout == null || account.userId != contactId]
[otherwise]
9 : <<create>>
10: doesContactExist(currentUserId, contactAccountCode)
11 : result
Contact entry already exists
12 : status code 401
alt
[result == true]
[result == false]
13 : if any error, show error and stop
14 : addContactToAddressBook(currentUserId, contactAccountCode)
15 : status code 200
16 : load and display success image
(for details on the load routine of the address book see "Home Page Load")

Auto complete suggestions



Press *ENTER* key in input capture



Alcuni link utili

- DAO, chisura dei preparedStatement e resultSet
  https://stackoverflow.com/questions/14546592/do-i-need-to-close-preparedstatement
- GET Generated Keys, estrarre le chiavi generate da una entry sul DB
  https://stackoverflow.com/questions/4224228/preparedstatement-with-statement-return-generated-keys
- GET parameters in thymeleaf
  https://stackoverflow.com/questions/39865482/how-can-i-call-getters-from-model-passed-to-thymeleaf-like-parameter
- Validating an email address
  https://stackoverflow.com/questions/201323/how-can-i-validate-an-email-address-using-a-regular-expression
  https://regexr.com
- Thymeleaf guide
  https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
- Altri link nel codice del progetto