Documentazione progetto TIW 2022

Alice Portentoso – 10664207 - 934939

1 – Versione Pure HTML

Data Requirements Analysis

(Entities, attributes, relationships)

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 **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.

Application Requirement Analysis

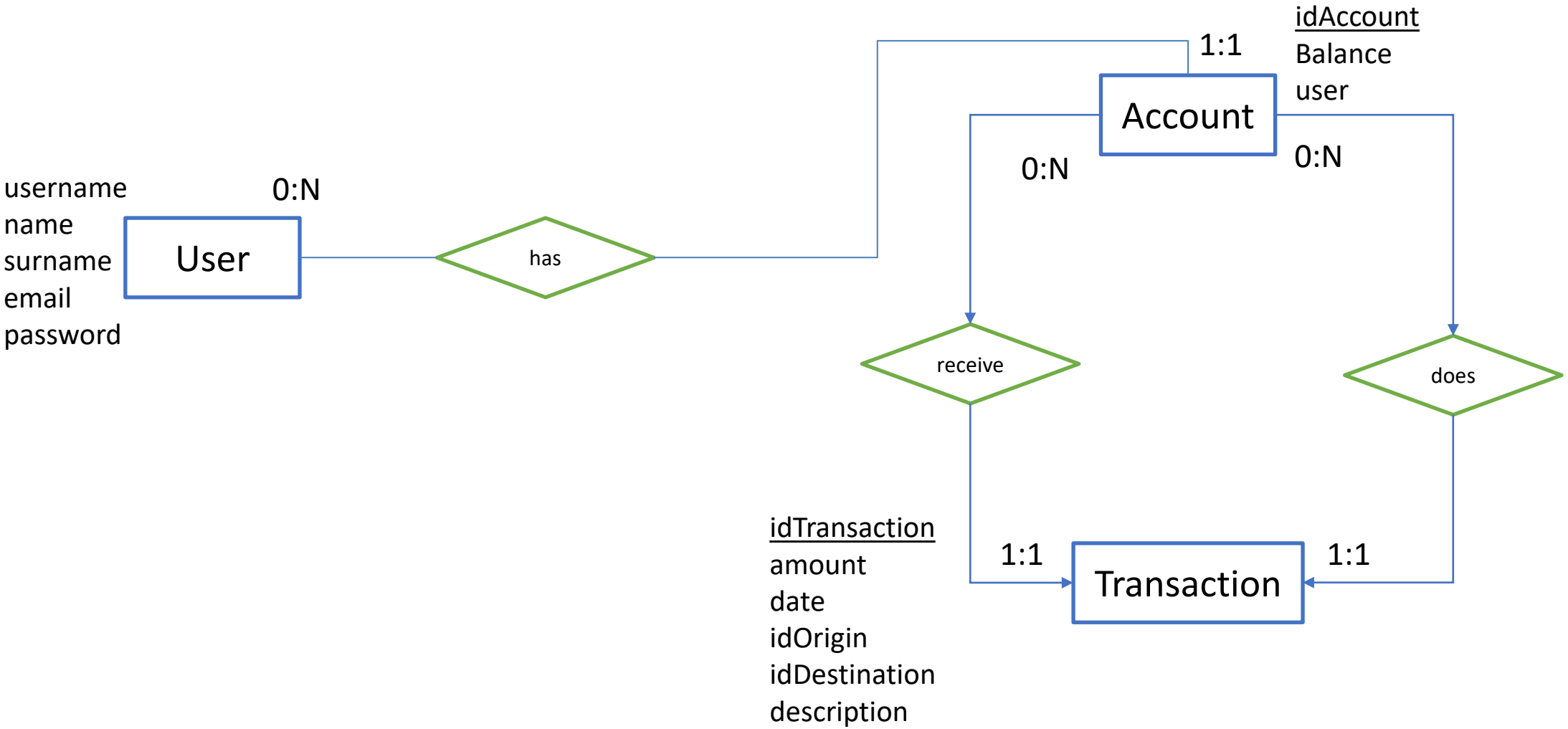
(Pages (views), view components, events, actions)

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 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.

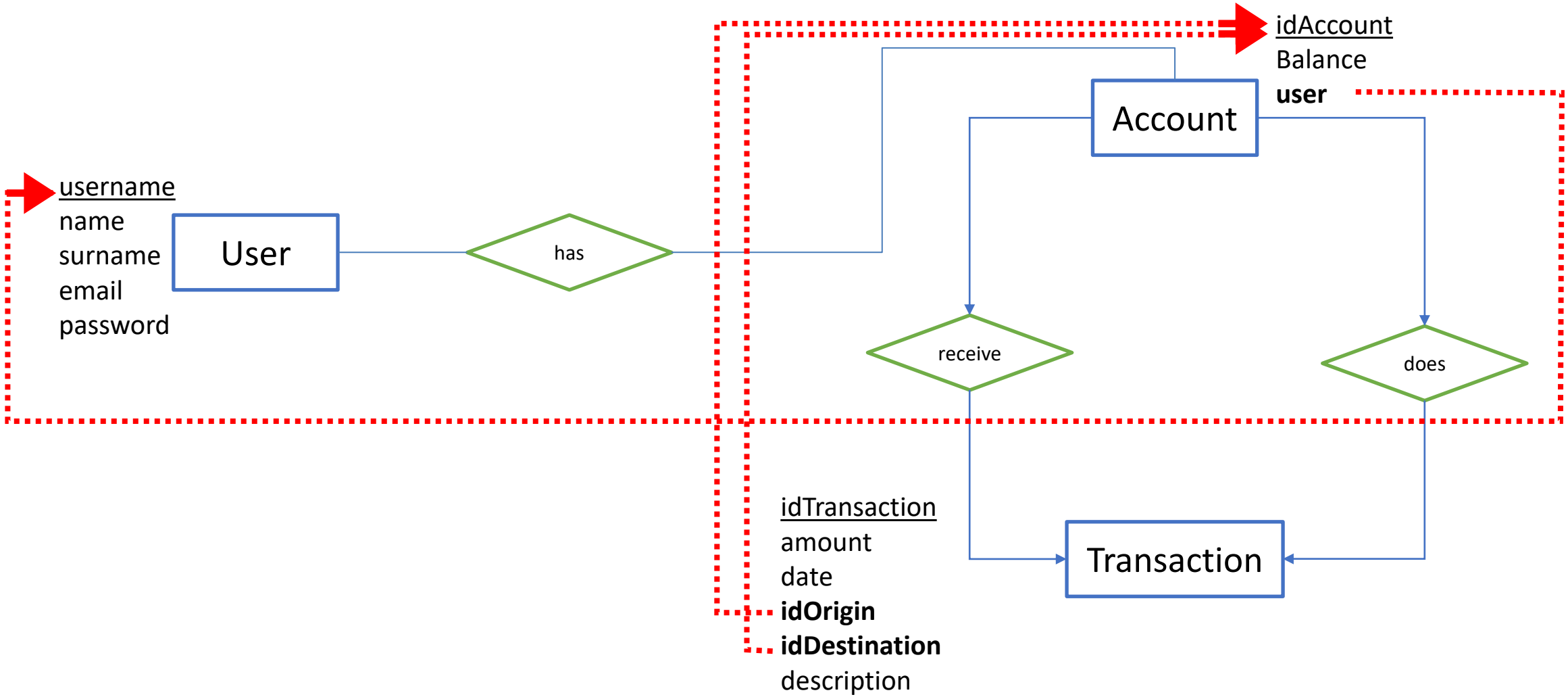
Scelte progettuali e completamento specifiche

- La pagina di default contiene sia il form per il login che per la registrazione
- Come codice utente è stato scelto lo username che deve essere unico
- Username e password rappresentano le credenziali per il login
- I codici dei conti sono numerici e unici
- Tutti i campi dei form sono obbligatori
- È possibile fare logout da ogni pagina
- Ogni pagina contiene un pulsante di Back che riporta alla pagina precedente: la pagina dei dettagli del conto riporta all'elenco dei conti e la pagina di esito del trasferimento riporta alla pagina dei dettagli del conto
- Se l'invio dei form di login e signup fallisce a causa dei campi inseriti l'errore viene mostrato nella stessa pagina
  - Il form di login fallisce se la coppia user-password non esiste nel db
  - Il form di registrazione fallisce se l'username scelto esiste già oppure se i campi «password» e «ripeti password» sono diversi
- Il form della nuova transazione mostra il motivo del fallimento in una nuova pagina
  - Il form di nuova transazione fallisce se il conto di destinazione non esiste, il conto di destinazione coincide con il conto sorgente, il conto di destinazione non appartiene all'utente selezionato, l'importo non è positivo o l'importo è maggiore del saldo sul conti

Database design



Database design - foreign key



Database design - DDL

```
CREATE TABLE `user` (
  `username` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`username`));
```

```
CREATE TABLE `account` (
  `idAccount` int NOT NULL AUTO_INCREMENT,
  `balance` decimal(19,4) NOT NULL,
  `user` varchar(45) NOT NULL,
  PRIMARY KEY (`idAccount`),
  KEY `username_idx` (`user`),
  CONSTRAINT `user` FOREIGN KEY (`user`)
    REFERENCES `user` (`username`));
```

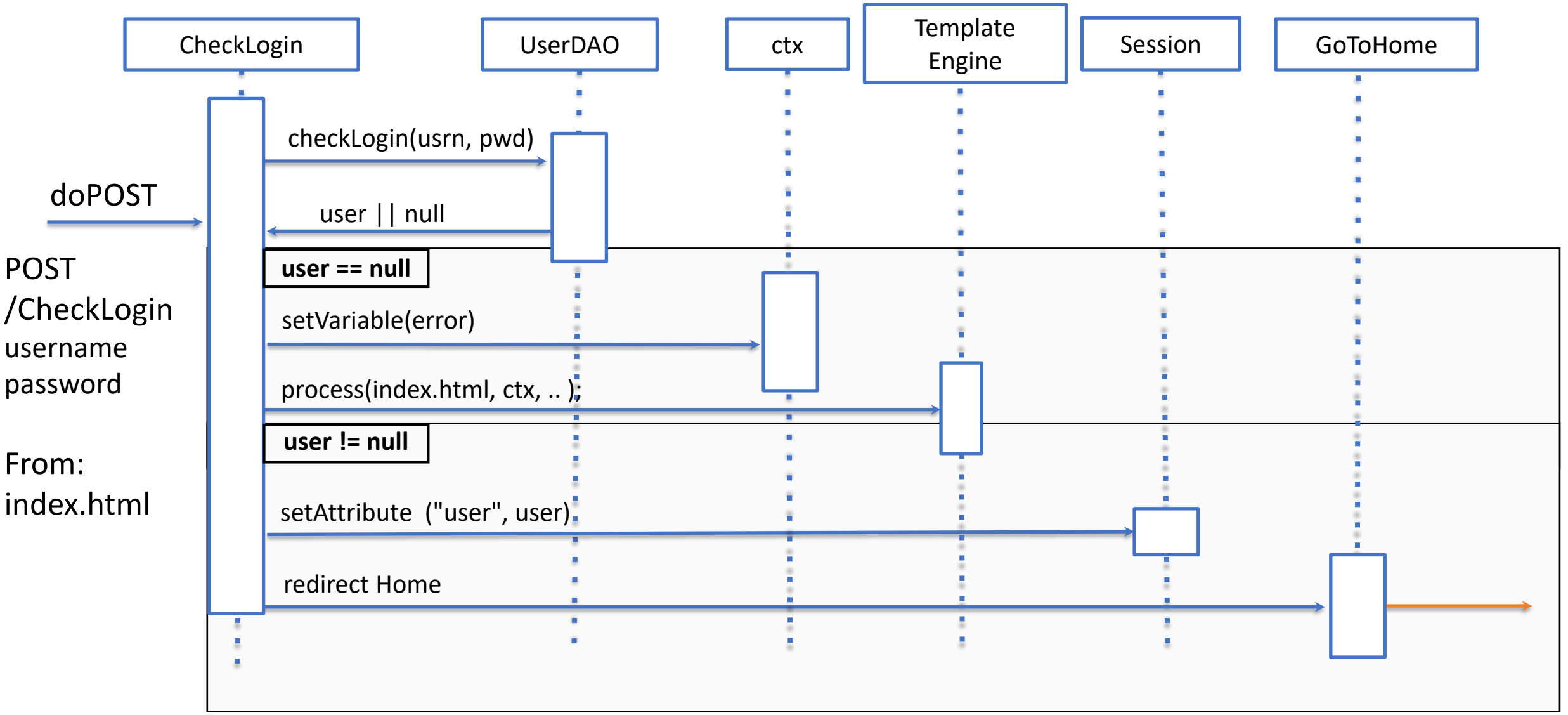
```
CREATE TABLE `transaction` (
  `idTransaction` int NOT NULL AUTO_INCREMENT,
  `amount` decimal(19,4) NOT NULL,
  `date` datetime NOT NULL,
  `idOrigin` int NOT NULL,
  `idDestination` int NOT NULL,
  `description` varchar(45) NOT NULL,
  PRIMARY KEY (`idTransaction`),
  KEY `idorigin_idx` (`idOrigin`),
  KEY `iddestination_idx` (`idDestination`),
  CONSTRAINT `iddestination` FOREIGN KEY (`idDestination`)
    REFERENCES `account` (`idAccount`),
  CONSTRAINT `idorigin` FOREIGN KEY (`idOrigin`)
    REFERENCES `account` (`idAccount`)) ;
```

Componenti

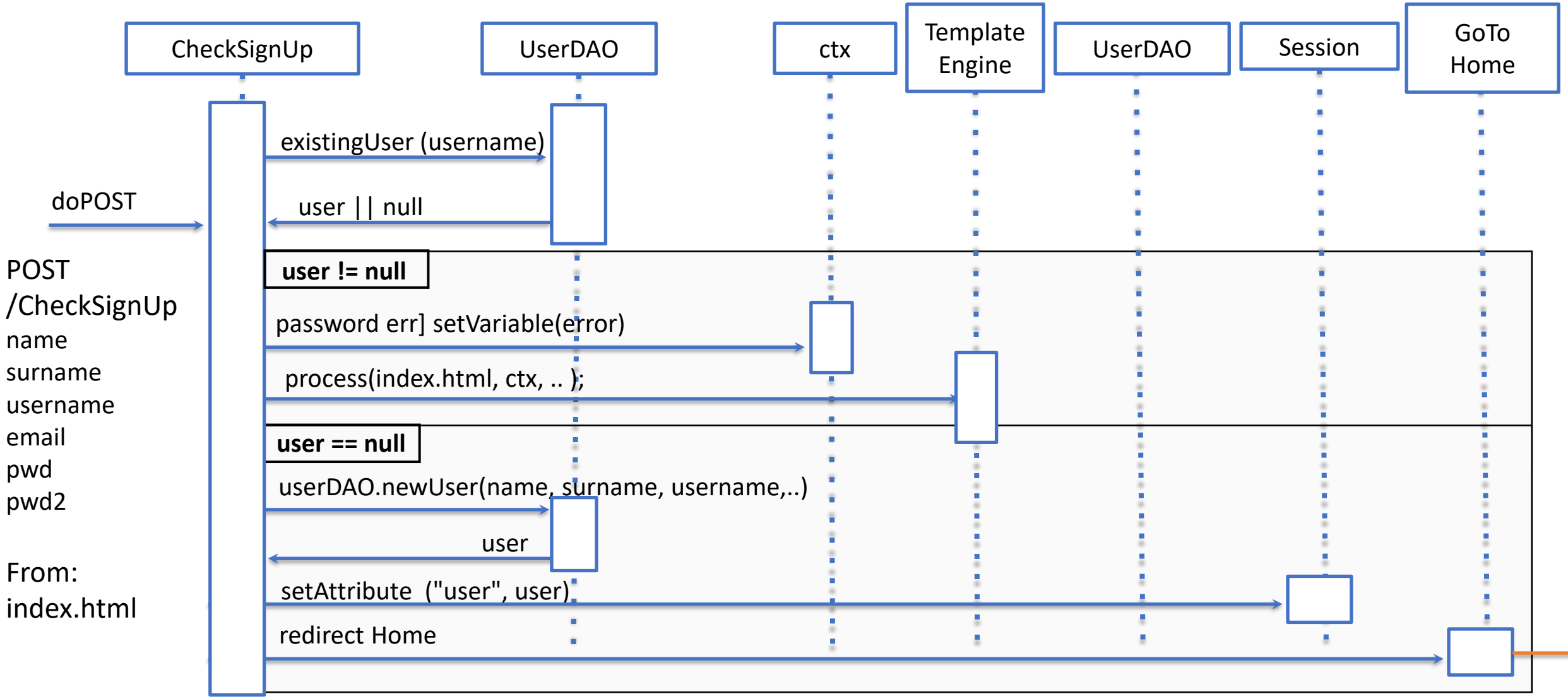
- **Beans**
  - User
  - Account
  - Transaction
- **Controllers**
  - CheckLogin
  - CheckSignup
  - GetAccountDetails
  - CreateTransaction
  - GoToHomePage
  - Logout
- **DAO**
  - UserDAO
    - checkLogin(username, password)
    - newUser(name, surname, username, email, password)
    - existingUser(username)
  - AccountDAO
    - findAccountsByUser(user)
    - findAccountsById(idAccount)
  - TransactionDAO
    - findTransactionsByAccount(idAccount)
    - createTransaction(idOrigin, userDestination, idDestination, amount, description, date)
- **View**
  - Index.html
  - Home.html
  - AccountDetails.html
  - FailedTransaction.html
  - TransactionConfirmation.html
- **Filter**
  - CheckLogin

Sequence Diagram – HTML Version

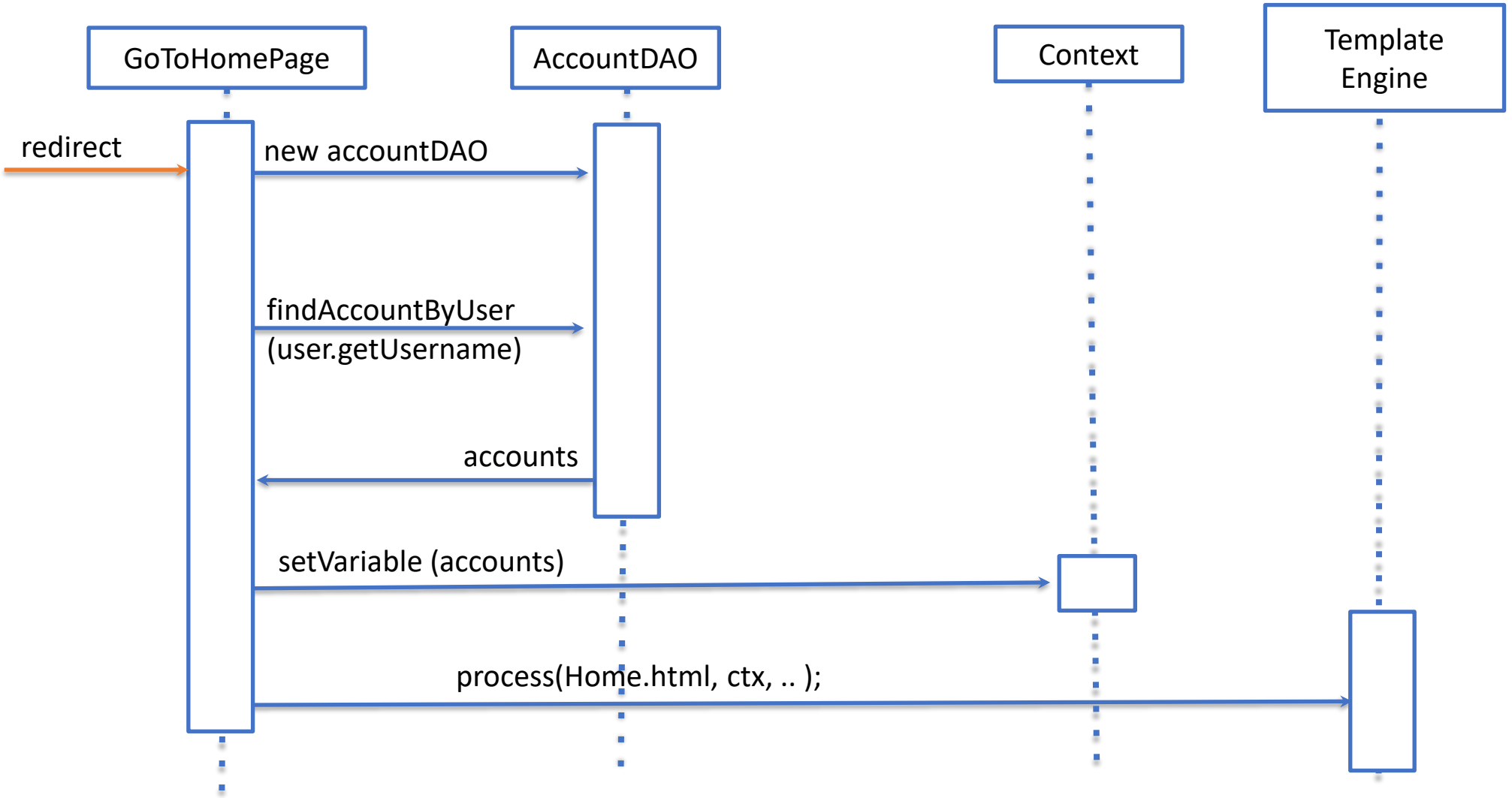
Event: login


CheckLogin
UserDAO
ctx
Template Engine
Session
GoToHome
doPOST
checkLogin(usrn, pwd)
user || null
user == null
setVariable(error)
process(index.html, ctx, ..);
user != null
setAttribute ("user", user)
redirect Home
POST
/CheckLogin
username
password
From:
index.html

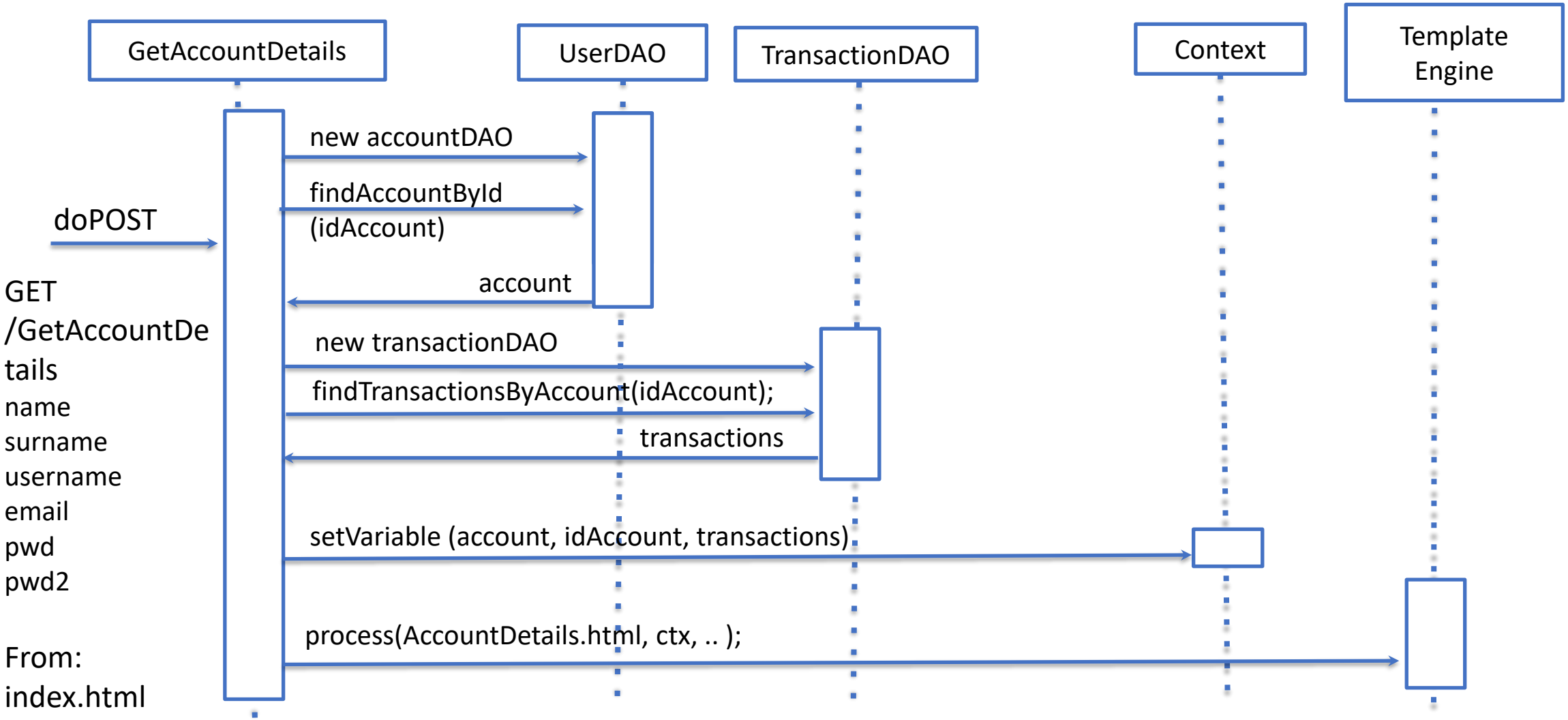
Event: signup



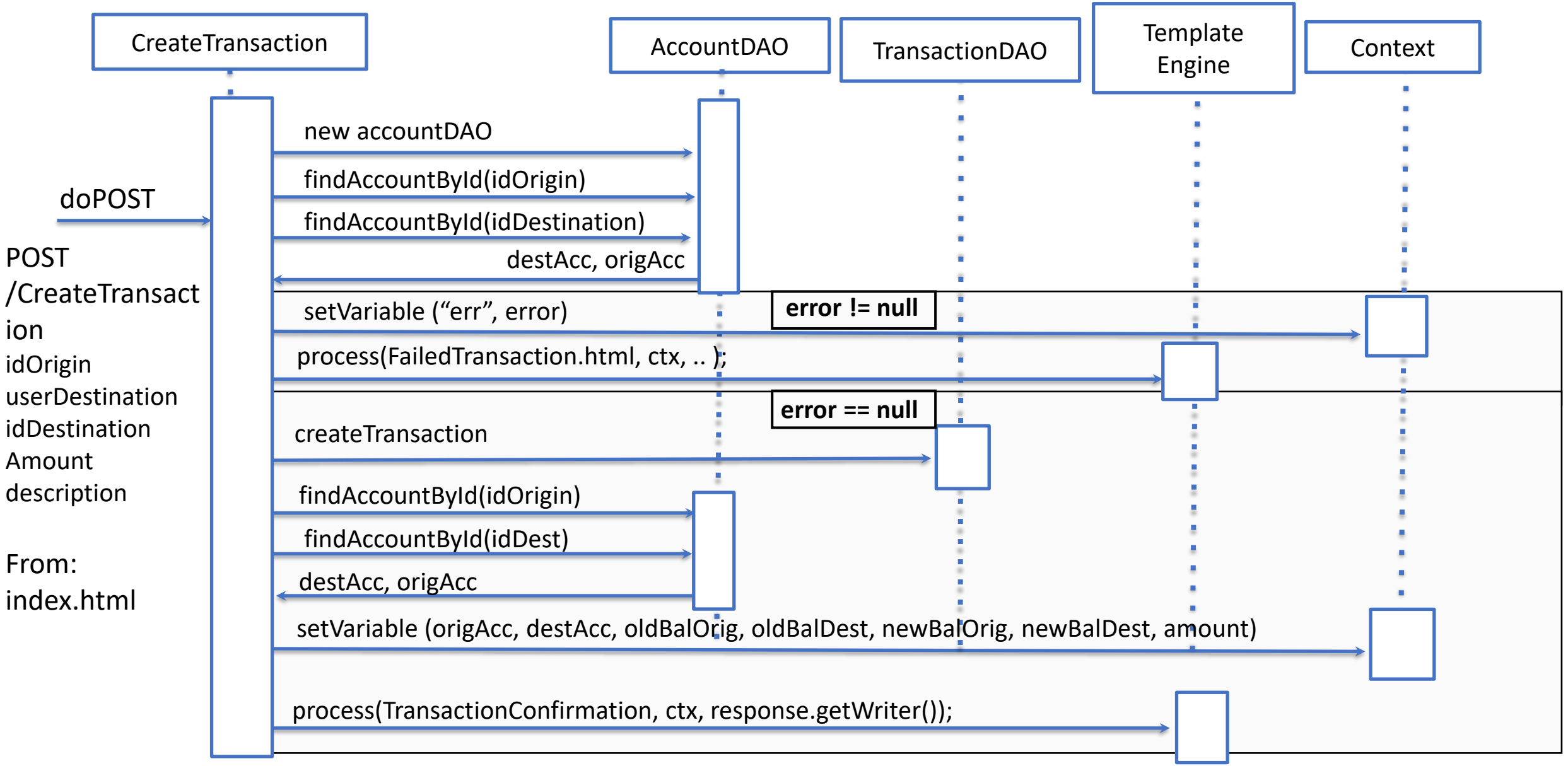
Event: go to home page



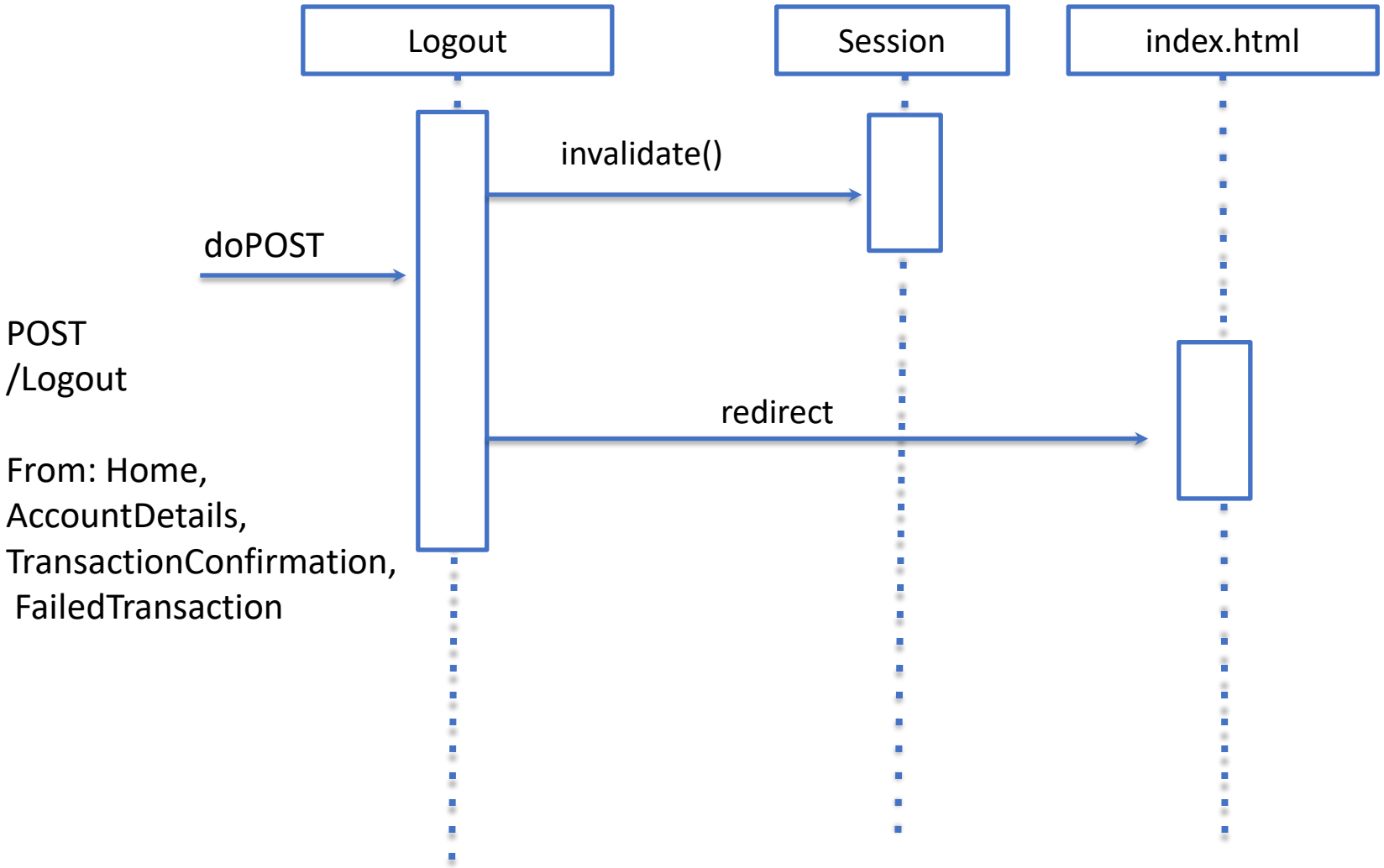
Event: select account



Event: create transaction

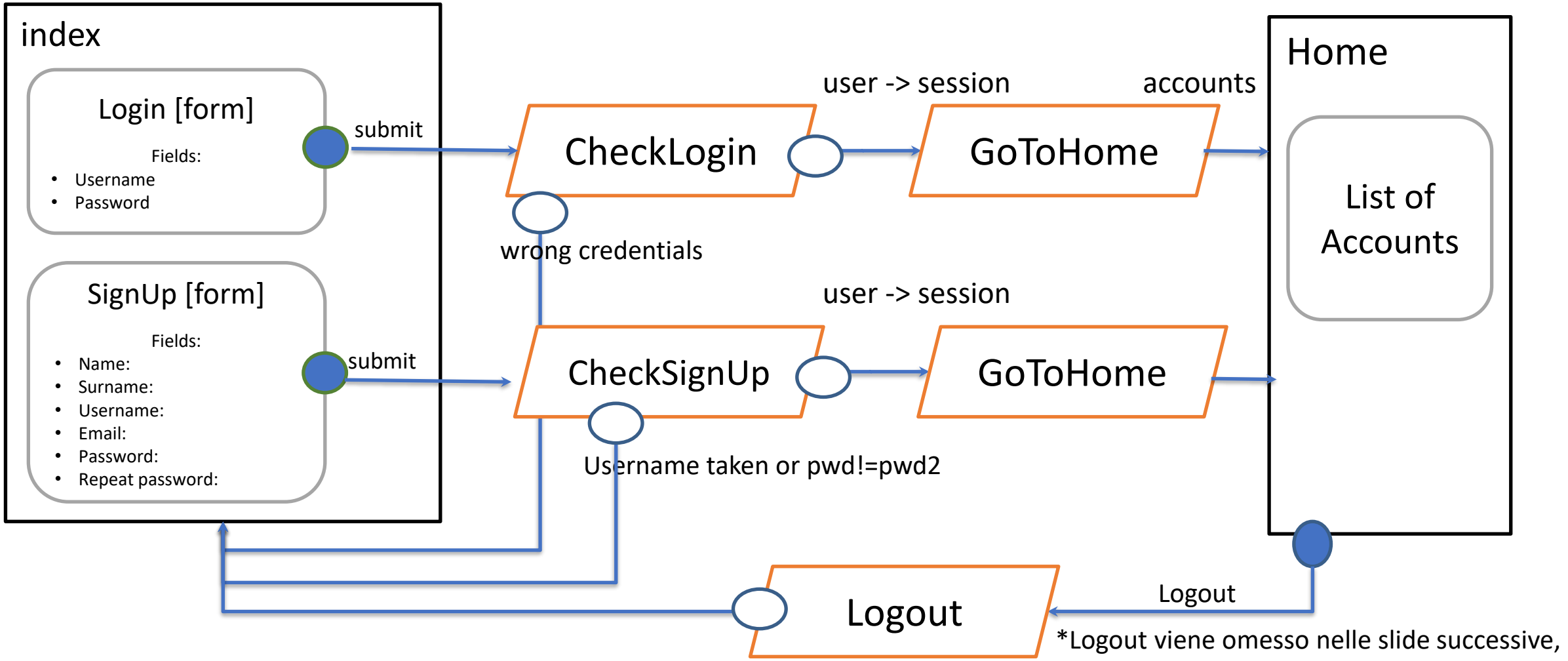


Event: logout

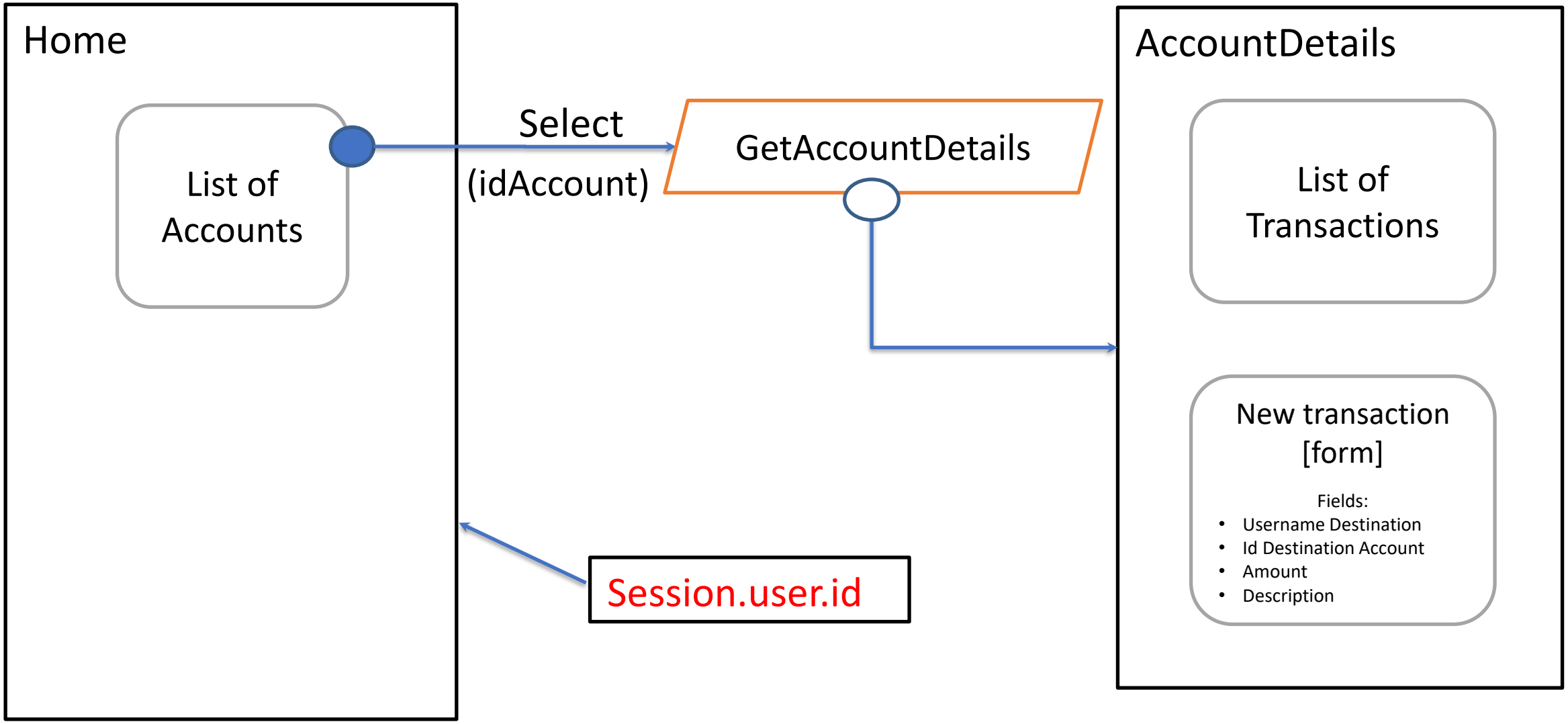


IFML – HTML Version

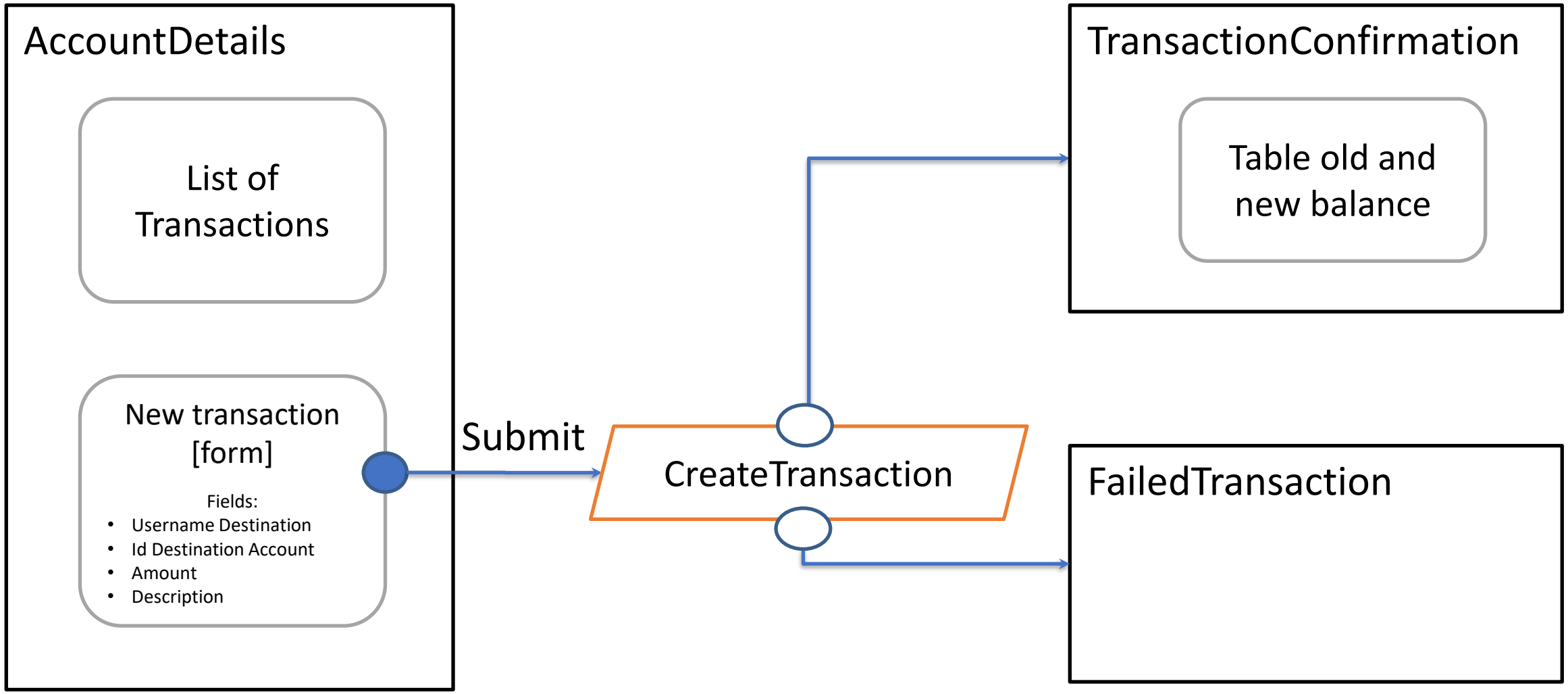
Login and Signup



Get account details



Create Transaction



Documentazione progetto TIW 2022

Alice Portentoso – 10664207 - 934939

2 – Versione RIA

Data Requirements Analysis

(Entities, attributes, relationships)

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 **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.

Application Requirement Analysis

(Pages (views), view components, events, actions)

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 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.

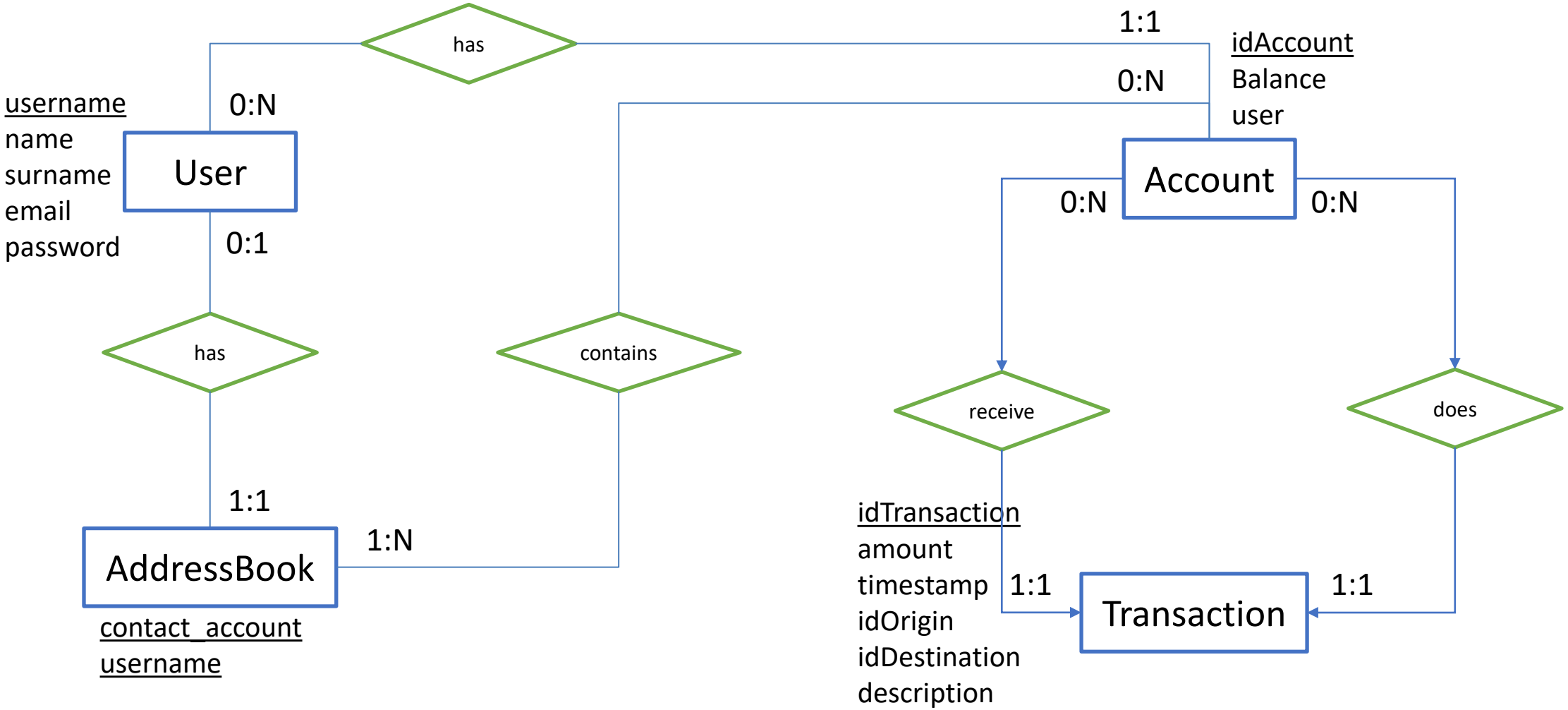
Completamento specifiche RIA

- - 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.

Scelte progettuali

- Come codice utente è stato scelto lo username che deve essere unico
- I codici dei conti sono numerici e unici
- Al caricamento della pagina viene selezionato con autoclick un account di cui mostrare già i dettagli
- Per la versione RIA viene aggiunta al DB l'entità AddressBook che mappa l'username corrente con i conti dei contatti salvati
- All'invio di una transazione l'applicazione controlla che il conto di origine sia diverso da quello di destinazione, che il conto appartenga all'utente specificato, che l'importo sia maggiore di 0 e non maggiore del saldo del conto, sia a lato client che a lato server
- Quando una transazione va a buon fine l'applicazione controlla se il contatto esiste già: se esiste compare solo il bottone di back, se non esiste chiede all'utente se vuole salvarlo
- Il form di un nuovo trasferimento consiglia i codici utenti salvati con un menù a tendina: una volta selezionato, nel campo conto destinazione verranno consigliati i suoi conti
- È possibile inviare un form premendo il tasto «invia» che simula un evento di click per il bottone

Database design



Database design - DDL

```
CREATE TABLE `user` (
  `username` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`username`));

CREATE TABLE `account` (
  `idAccount` int NOT NULL
    AUTO_INCREMENT,
  `balance` decimal(19,4) NOT NULL,
  `user` varchar(45) NOT NULL,
  PRIMARY KEY (`idAccount`),
  KEY `username_idx` (`user`),
  CONSTRAINT `user` FOREIGN KEY (`user`)
   REFERENCES `user` (`username`));
```

```
CREATE TABLE `transaction` (
  `idTransaction` int NOT NULL
     AUTO_INCREMENT,
  `amount` decimal(19,4) NOT NULL,
  `date` datetime NOT NULL,
  `idOrigin` int NOT NULL,
  `idDestination` int NOT NULL,
  `description` varchar(45) NOT NULL,
  PRIMARY KEY (`idTransaction`),
  KEY `idorigin_idx` (`idOrigin`),
  KEY `iddestination_idx` (`idDestination`),
  CONSTRAINT `iddestination`
FOREIGN KEY (`idDestination`)
    REFERENCES `account` (`idAccount`),
  CONSTRAINT `idorigin`
FOREIGN KEY (`idOrigin`)
    REFERENCES `account` (`idAccount`)) ;
```

```
CREATE TABLE `address_book` (
`username` varchar(45) NOT NULL,
`contact_account` int NOT NULL,
PRIMARY KEY (`username`,
`contact_account`),
KEY `idAccount_idx` (`contact_account`),
KEY `username_idx` (`username`),
CONSTRAINT `username`
FOREIGN KEY (`username`)
  REFERENCES `user` (`username`));
```

Componenti Server Side

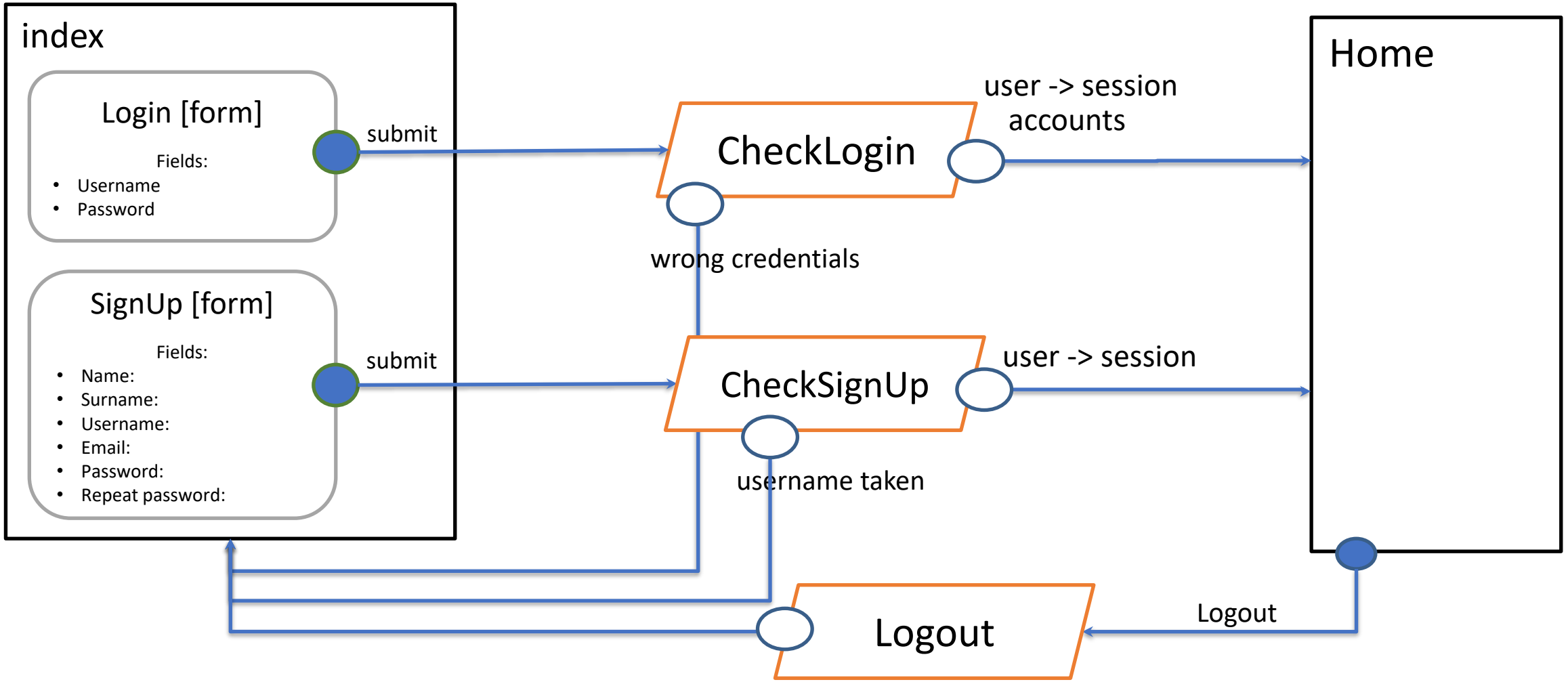
- **Beans**
  - User
  - Account
  - Transaction
  - AddressBook
- **Controllers**
  - CheckLogin
  - CheckSignup
  - GetAccountsList
  - GetAccountDetails
  - CreateTransaction
  - GetAddressBook
  - AddContact
  - Logout
- **DAO**
  - UserDAO
    - checkLogin(username, password)
    - getUserByUsername(username)
    - newUser(name, surname, username, email, password)
  - AccountDAO
    - findAccountsByUser(user)
    - findAccountsById(idAccount)
  - TransactionDAO
    - findTransactionsByAccount(idAccount)
    - createTransaction(idOrigin, userDestination, idDestination, amount, description, date)
  - AddressBookDAO
    - findAddressBookByUsername(username)
    - existsContact(username, contactAccount)
    - addContactToAddressBook(username, contactAccount)
- **View**
  - Index.html
  - Home.html
- **Filter**
  - CheckLogin

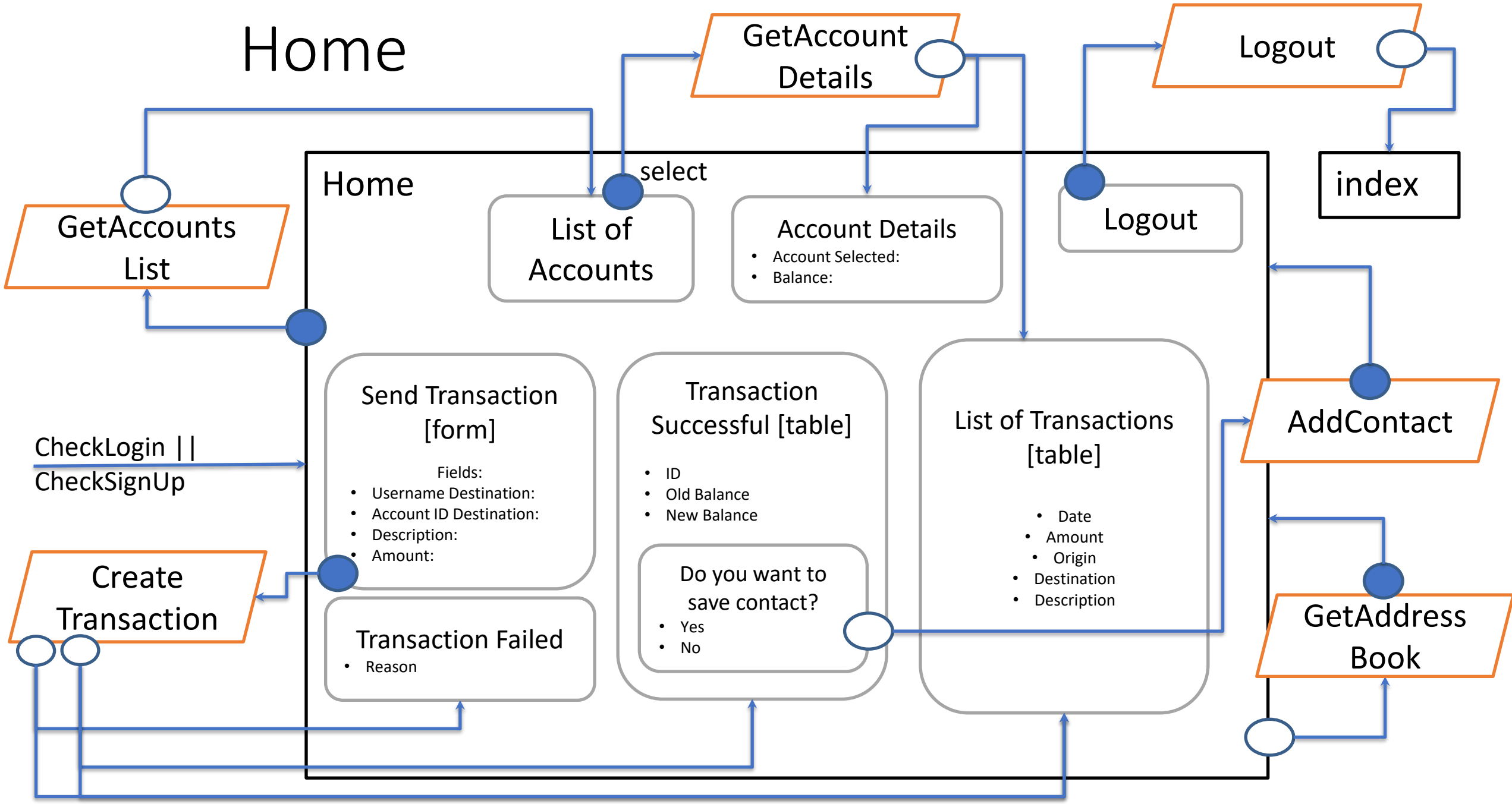
Componenti Client Side

- **Index.html + login.js**
  - LoginForm
  - SignupForm
- **Home.html + home.js**
  - pageOrchestrator
    - start : crea gli elementi della pagina
    - refresh : mostra e aggiorna gli elementi
  - welcome
    - show : mostra il messaggio di benvenuto
  - accountsList
    - show : richiede al server la lista dei conti
    - create_account_button : appende al documento un bottone per ogni conto trovato
    - autoclick : simula un click per un conto
  - transactionConfirmation
    - show : crea la tabella che mostra I vecchi e nuovi saldi
  - transactionFailed
    - show : mostra la ragione del fallimento
  - accountDetails
    - show : richiede al server i dettagli del conto e la lista delle transazioni
    - create_transactions_table : crea e riempie la tabella delle transazioni
    - create_transaction : controlla l'appartenenza del conto destinazione, controlla che l'import non sia maggiore del saldo e che sia positivo. Se corretti, manda i dati al server
  - addressBook
    - load : chiede al server i propri contatti salvati in rubrica
    - autoCompleteUsername : suggerisce in un menu a tendina gli username tra quelli salvati in rubrica
    - autoCompleteAccount : suggerisce i conti tra quelli, salvati in rubrica, dello username selezionato
    - addContact : invia al server i dati per inserire in rubrica un nuovo contatto
    - checkExistingContact : controlla se il contatto esiste già in rubrica, per non chiedere se vuole salvarlo

IFML – HTML Version

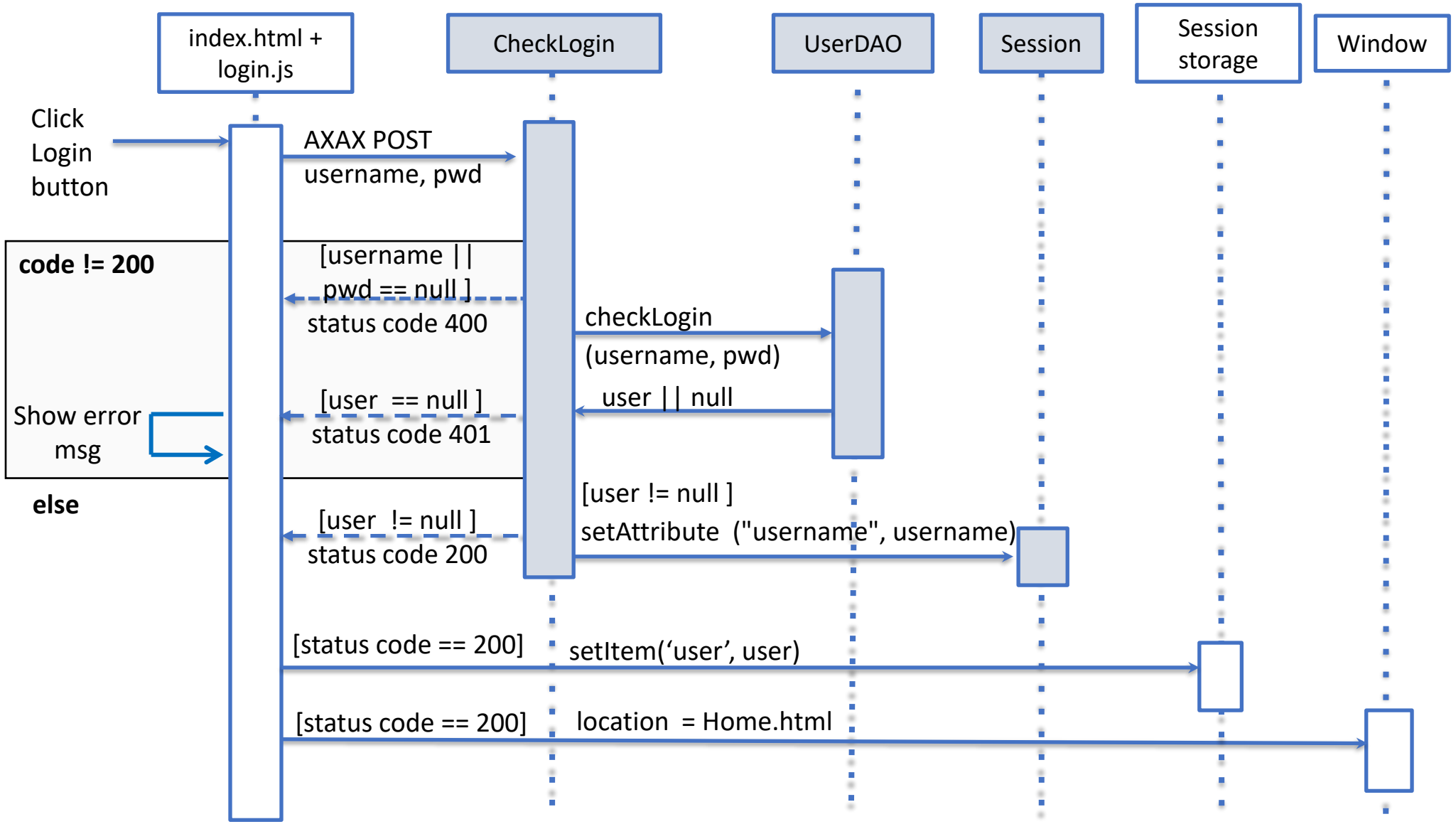
Login and Signup (index)



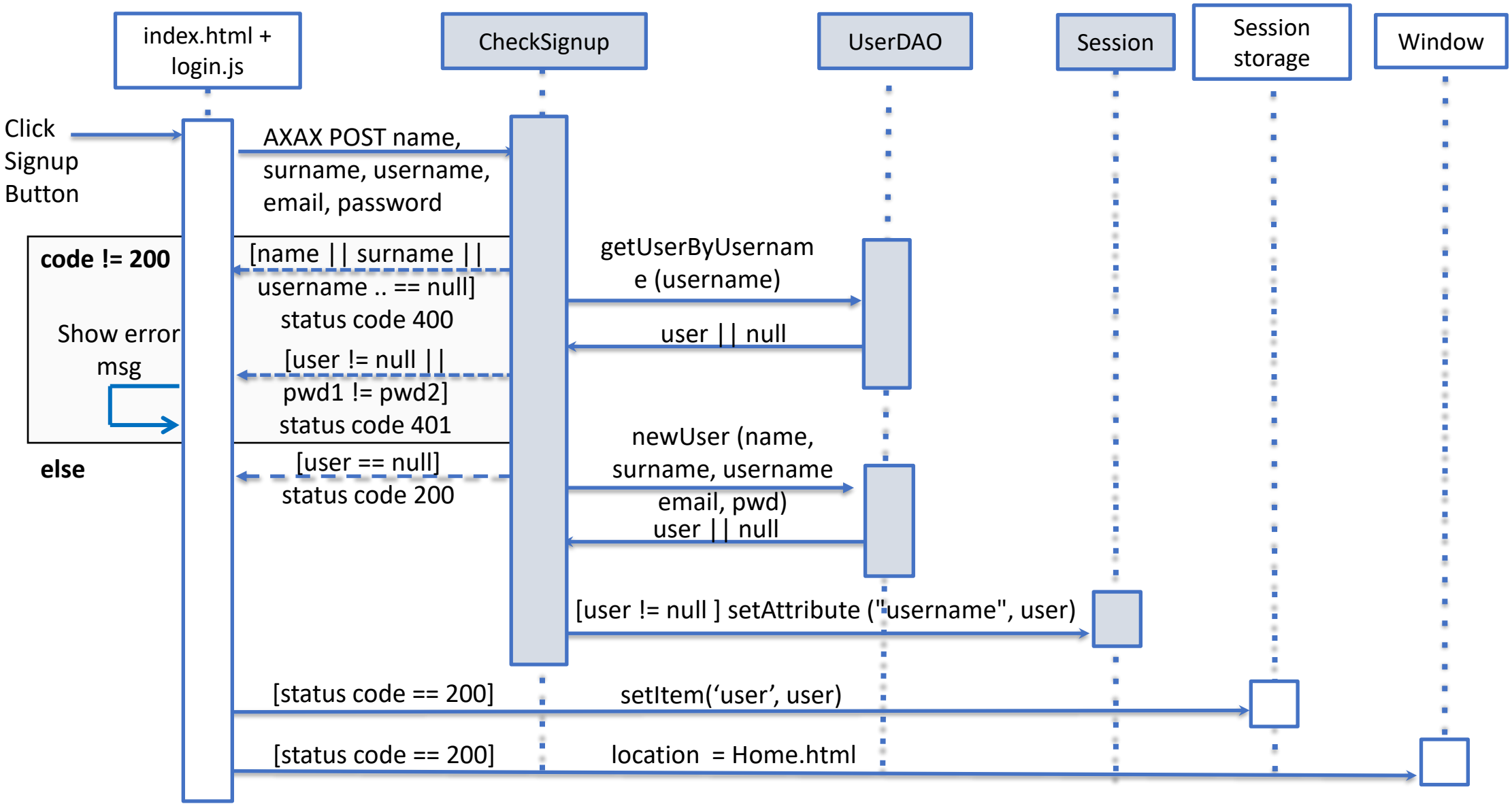

Home
GetAccount Details
Logout
Home
select
index
GetAccounts List
List of Accounts
Account Details
Account Selected:
Balance:
Logout
Send Transaction [form]
Fields:
Username Destination:
Account ID Destination:
Description:
Amount:
Transaction Successful [table]
ID
Old Balance
New Balance
List of Transactions [table]
Date
Amount
Origin
Destination
Description
AddContact
CheckLogin ||
CheckSignUp
Create Transaction
Transaction Failed
Reason
Do you want to save contact?
Yes
No
GetAddress Book

Sequence Diagram – RIA Version

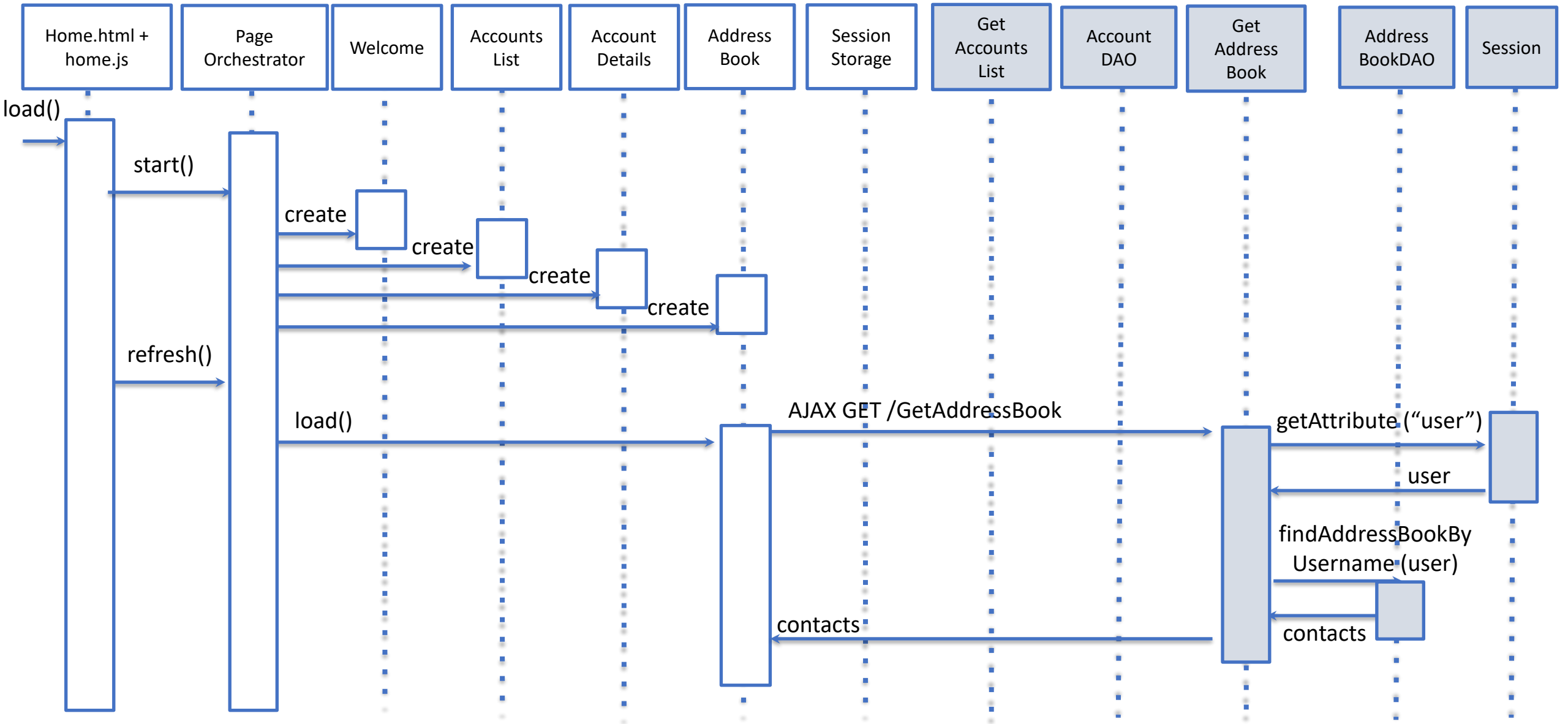
Evento: login



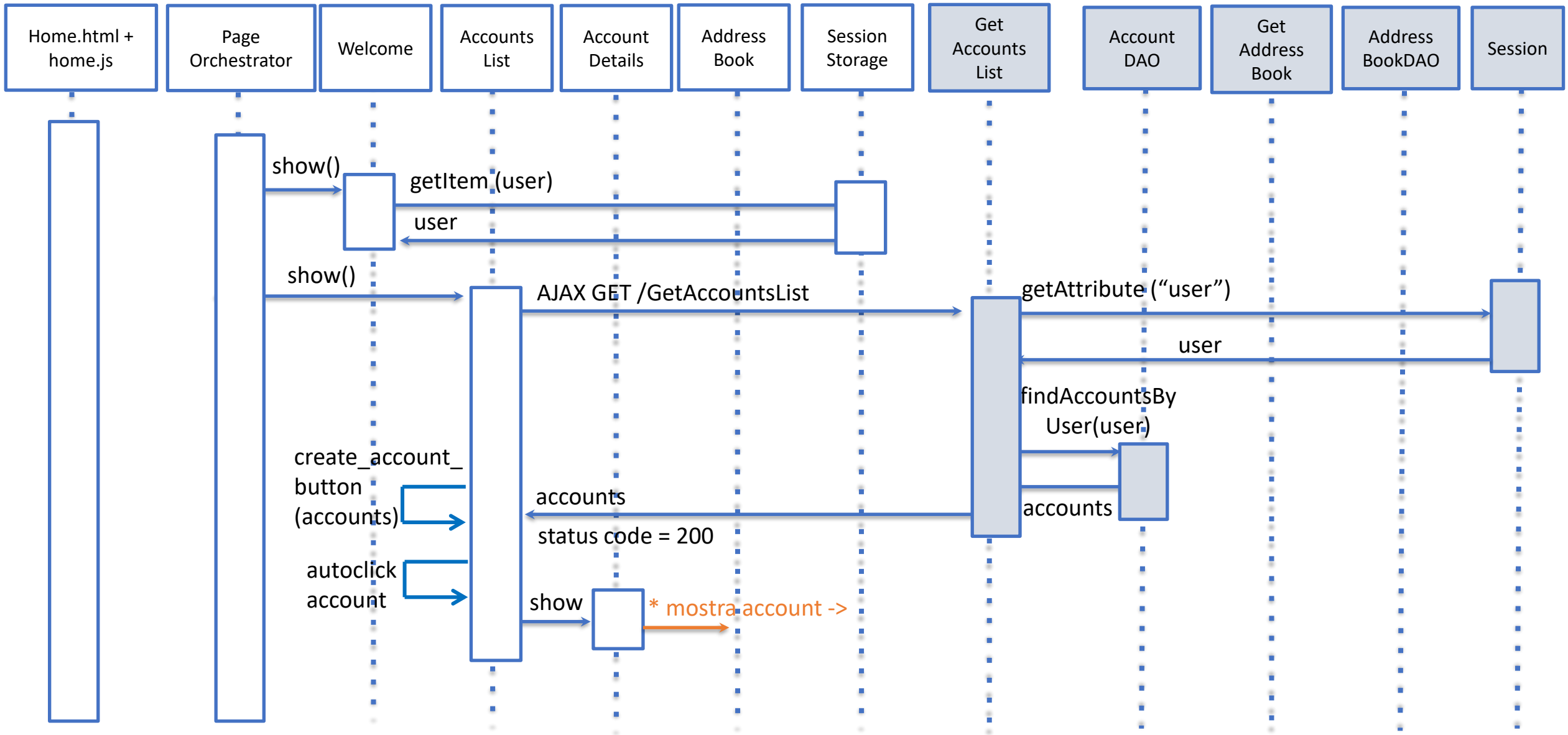
Evento: signup



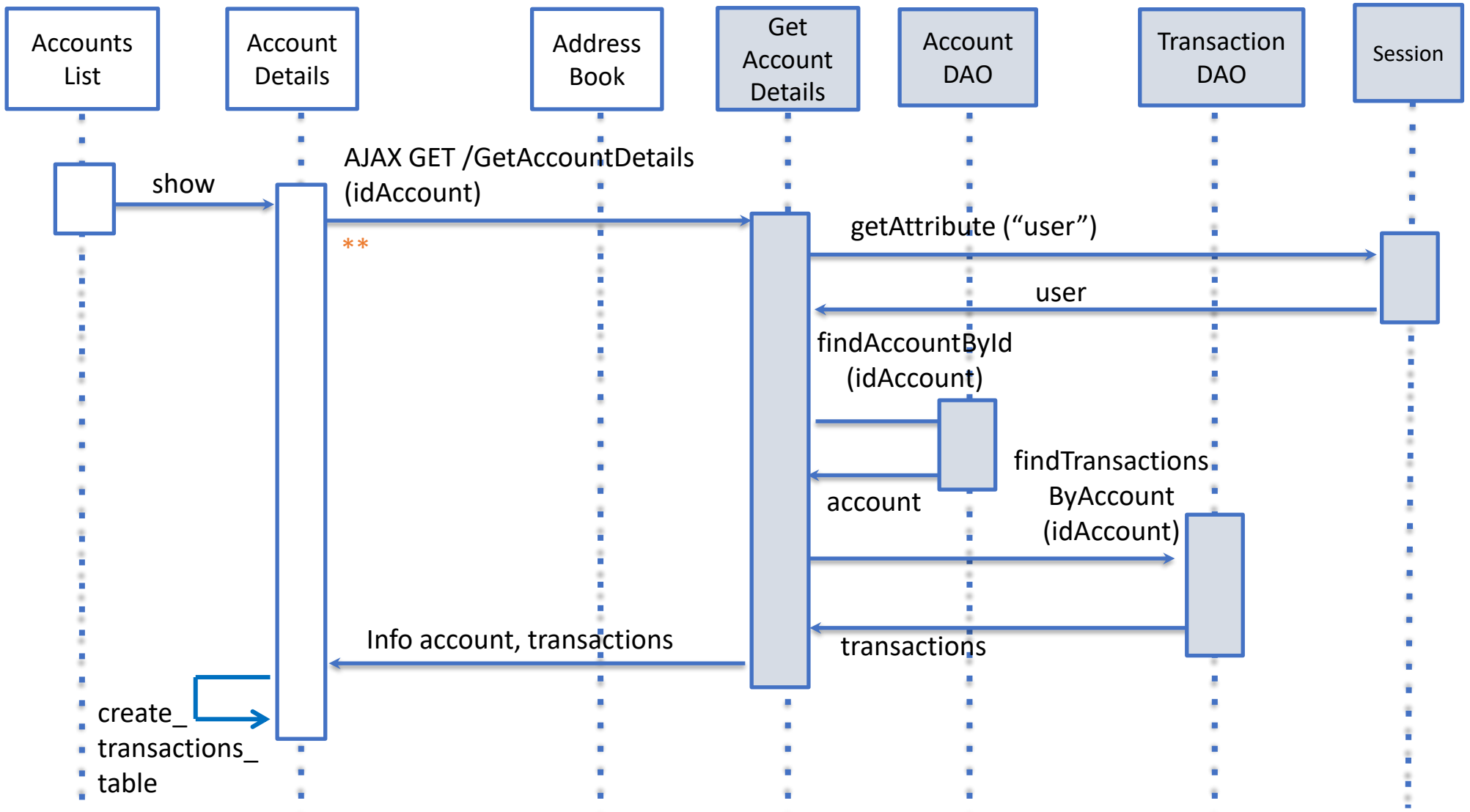
Evento: caricamento Home Page 1



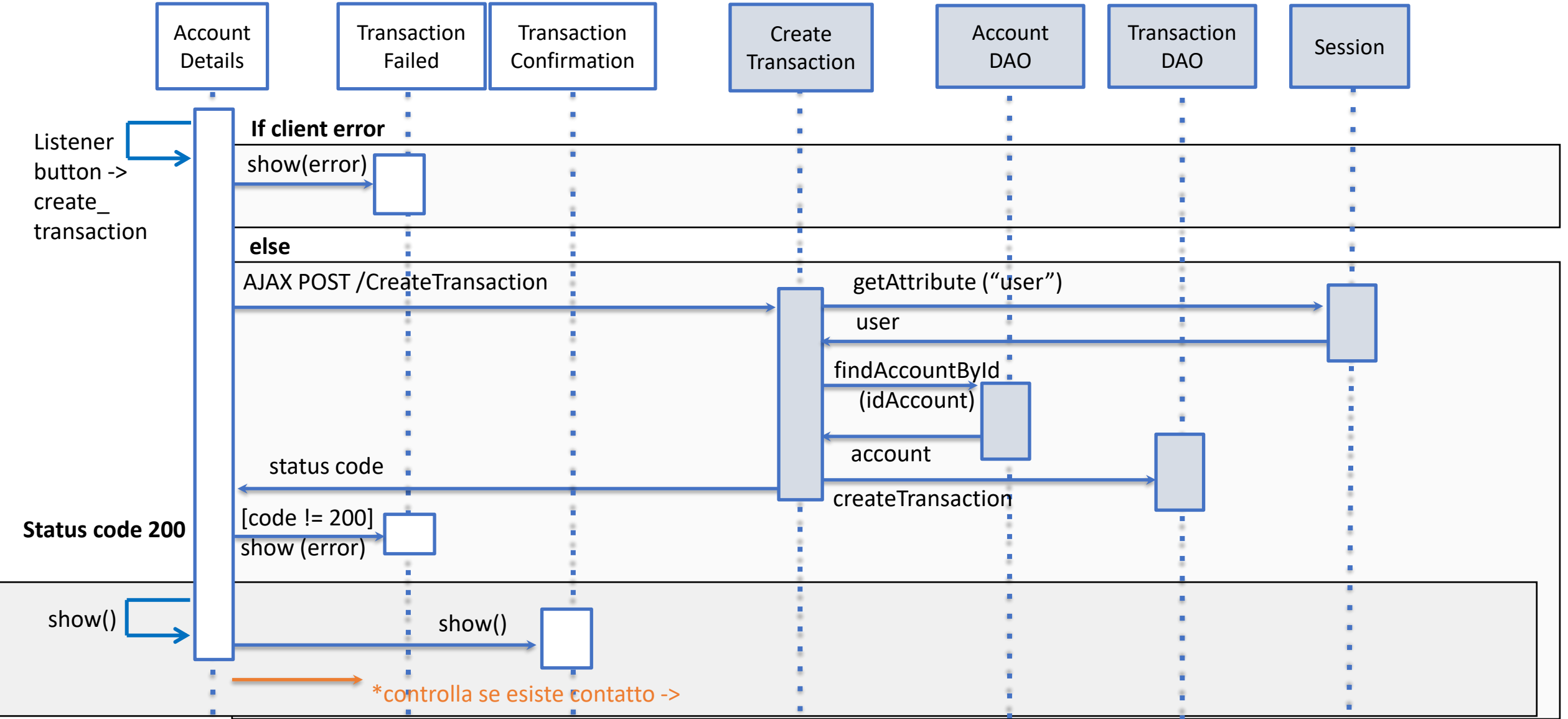
Evento: caricamento Home Page 2



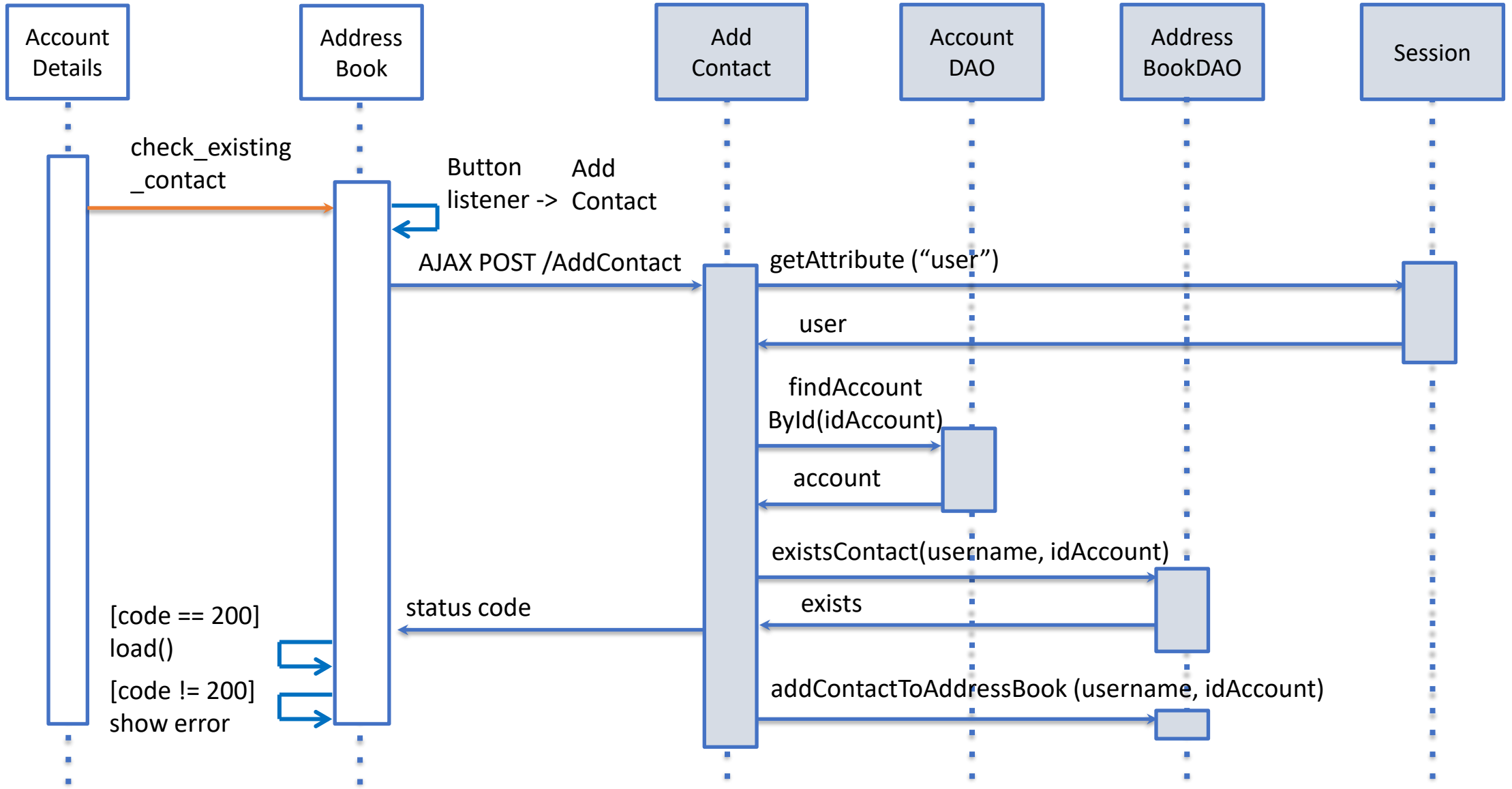
Evento: select account



Evento: new transaction



Evento: new contact



Evento: logout