Money Transfer Project

Web Technologies

Alberto Latino

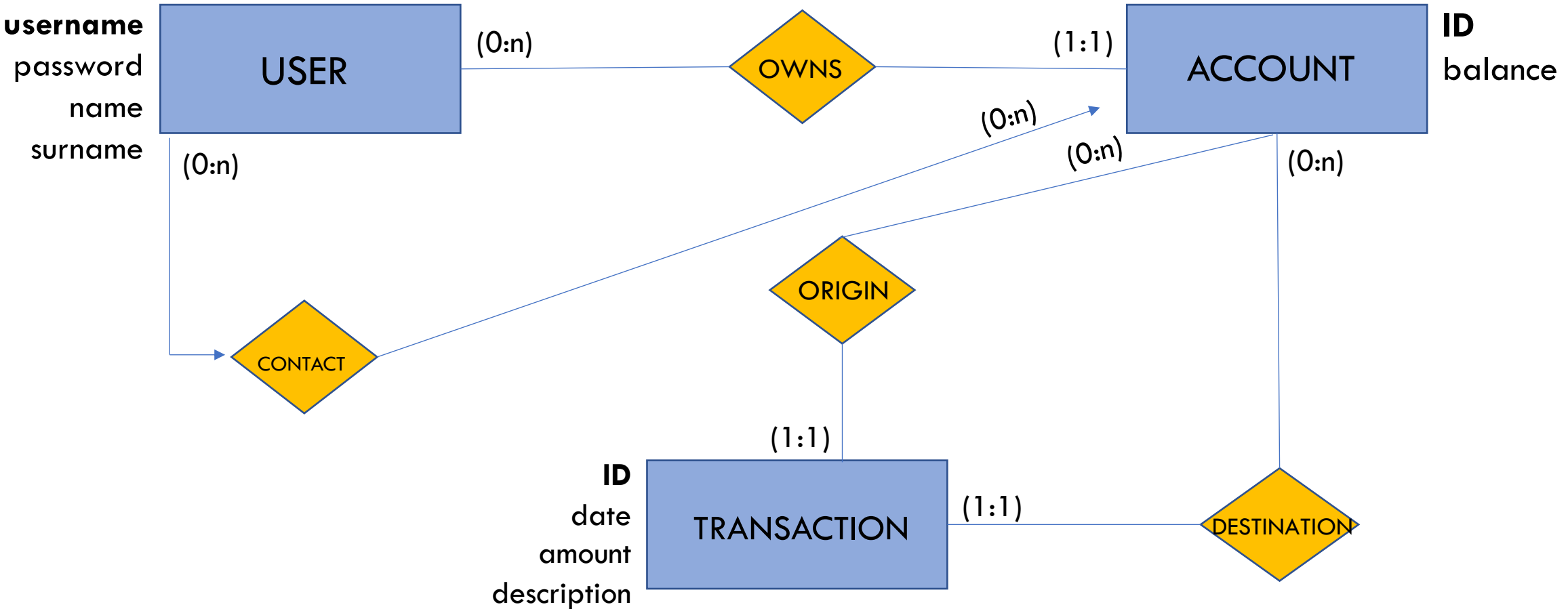
Francesco Gonzales

Data Analysis

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. Un utente ha un nome, un codice 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. In caso di verifica di entrambe le condizioni, l'applicazione deduce l'importo dal conto origine, aggiunge l'importo al conto destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati del conto di origine e destinazione, con i rispettivi saldi aggiornati. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato il conto di origine deve essere accreditato e viceversa.

Entities, **attributes**, **relationships**

Database Design



Application Requirement Analysis

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. Un utente ha un nome, un codice 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. In caso di verifica di entrambe le condizioni, l'applicazione deduce l'importo dal conto origine, aggiunge l'importo al conto destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati del conto di origine e destinazione, con i rispettivi saldi aggiornati. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato il conto di origine deve essere accreditato e viceversa.

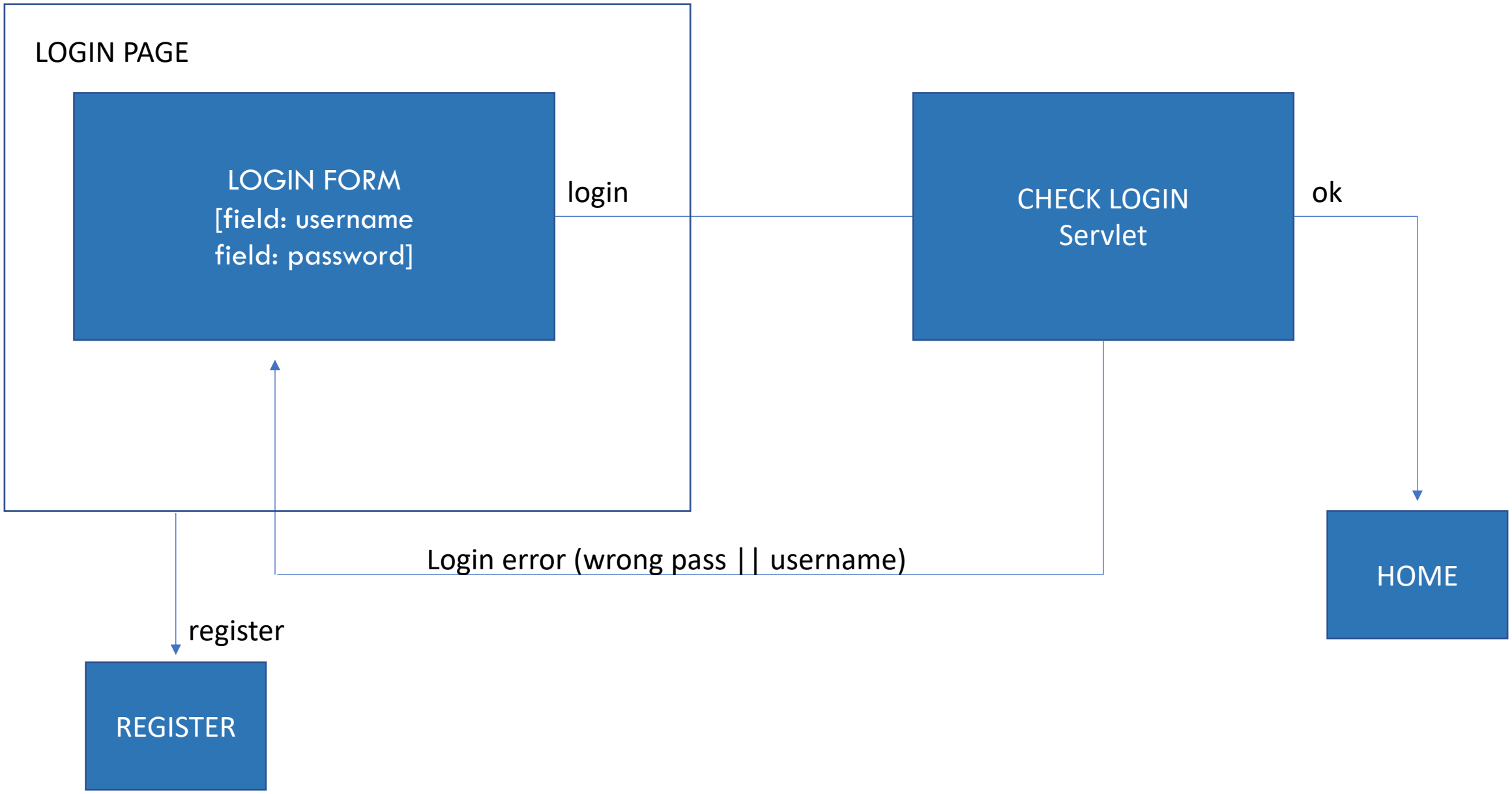
Pages (views), **view components**, **events**, **actions**

Application Requirements Analysis

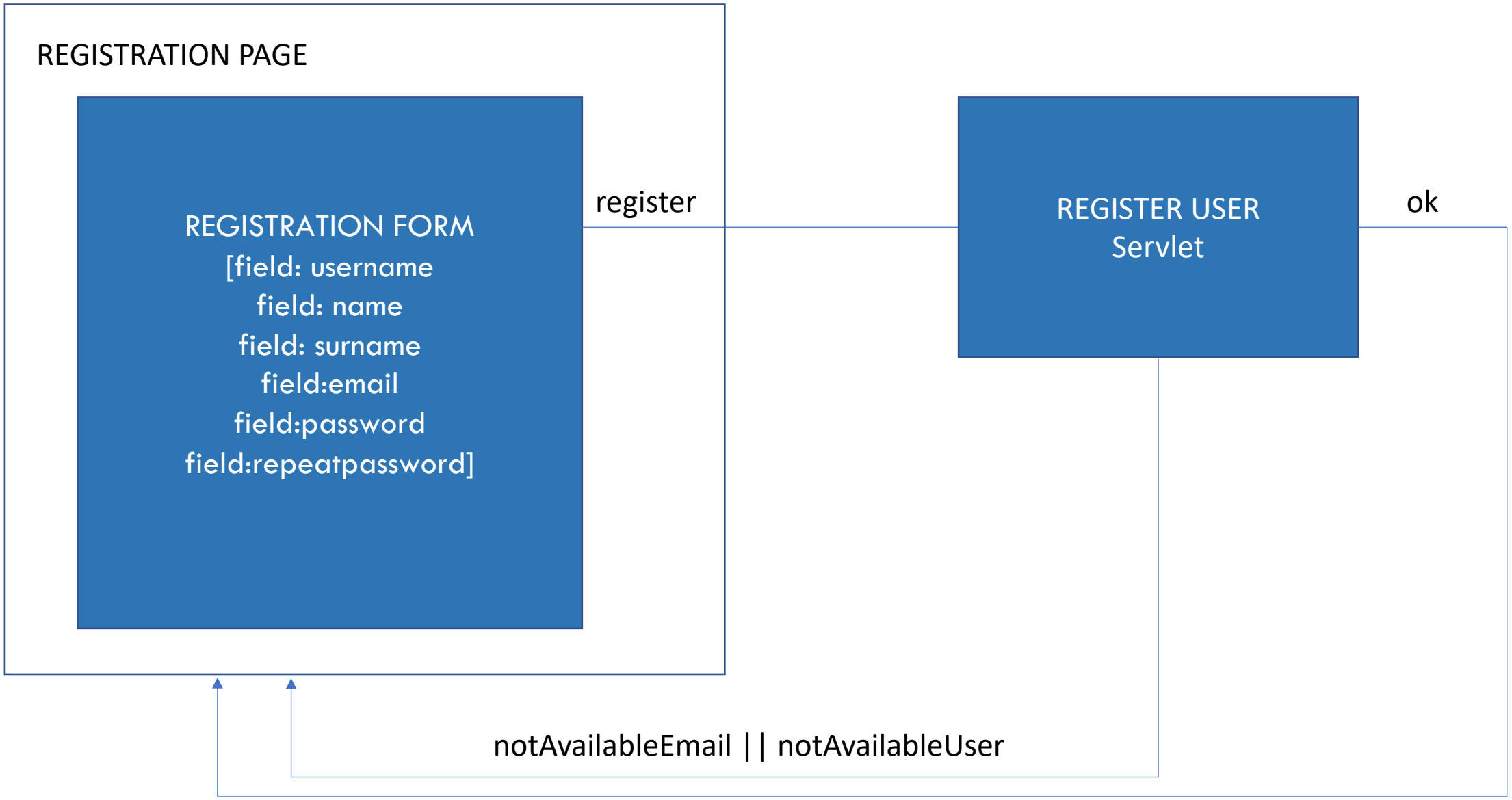
• L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password", anche a lato client. La registrazione controlla l'unicità dello username.
• 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 destinatario.

Pages (views), view components, events, actions

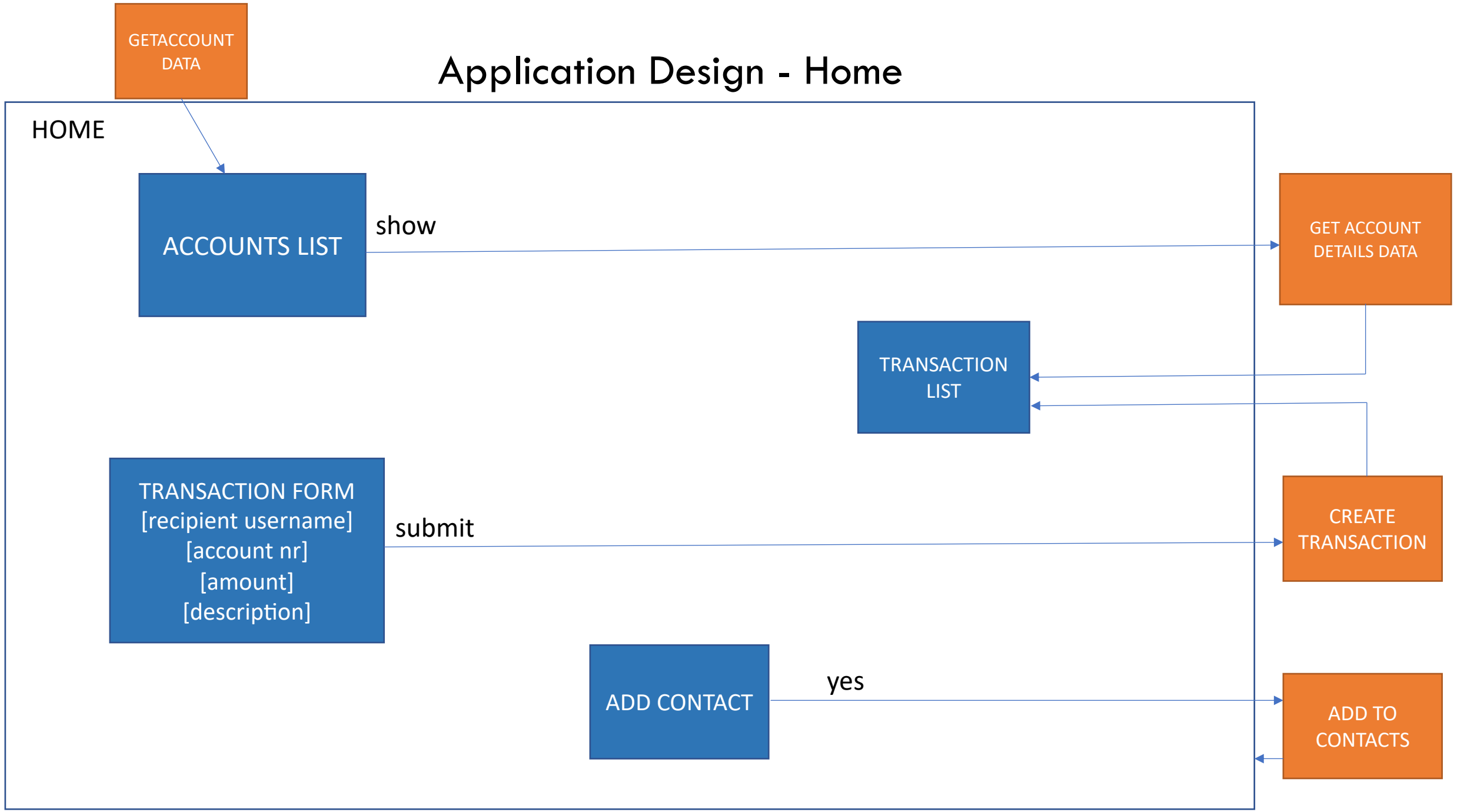
Application Design - Login



Application Design - Registration



Application Design - Home



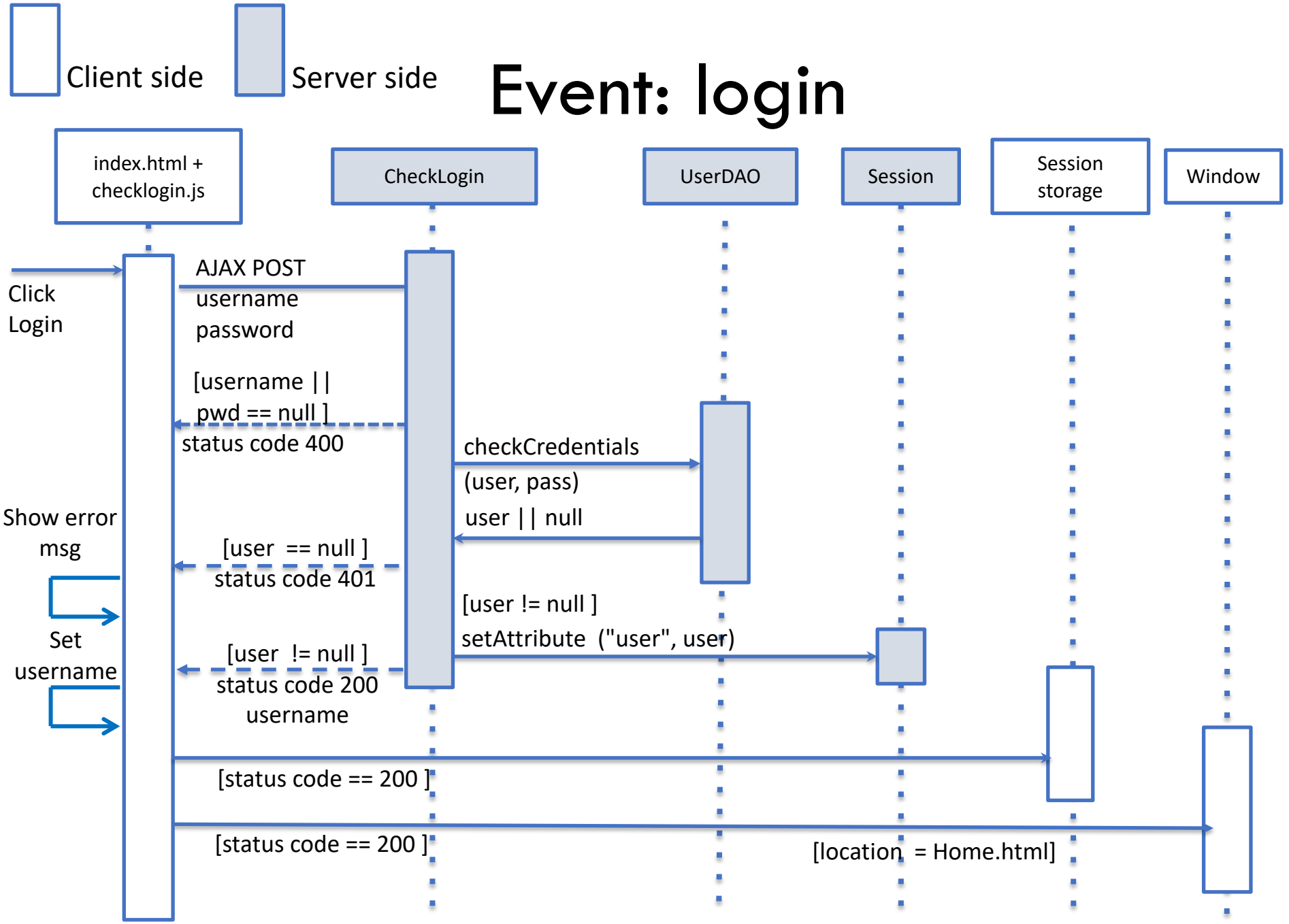
Events and Actions

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
index → login form → login	Data check	POST (username password)	Check Credentials
Home → load	Update view with account data and transactions	GET ()	Find User Accounts
Home →show Account Details	Update view with transactions of te selected account	GET(id account)	Find transactions by account
Home →accounts list	Sort	-	-
Wizard → next, prev, cancel	Data check, change fieldset	-	-
Wizard → submit	Amount check	POST (transaction data)	Checks & transaction creation
Registration->registrationForm->register	Data check(password fields and email)	POST(registration data)	Checks, existence of username, email checks, password.equals (repeatpass)

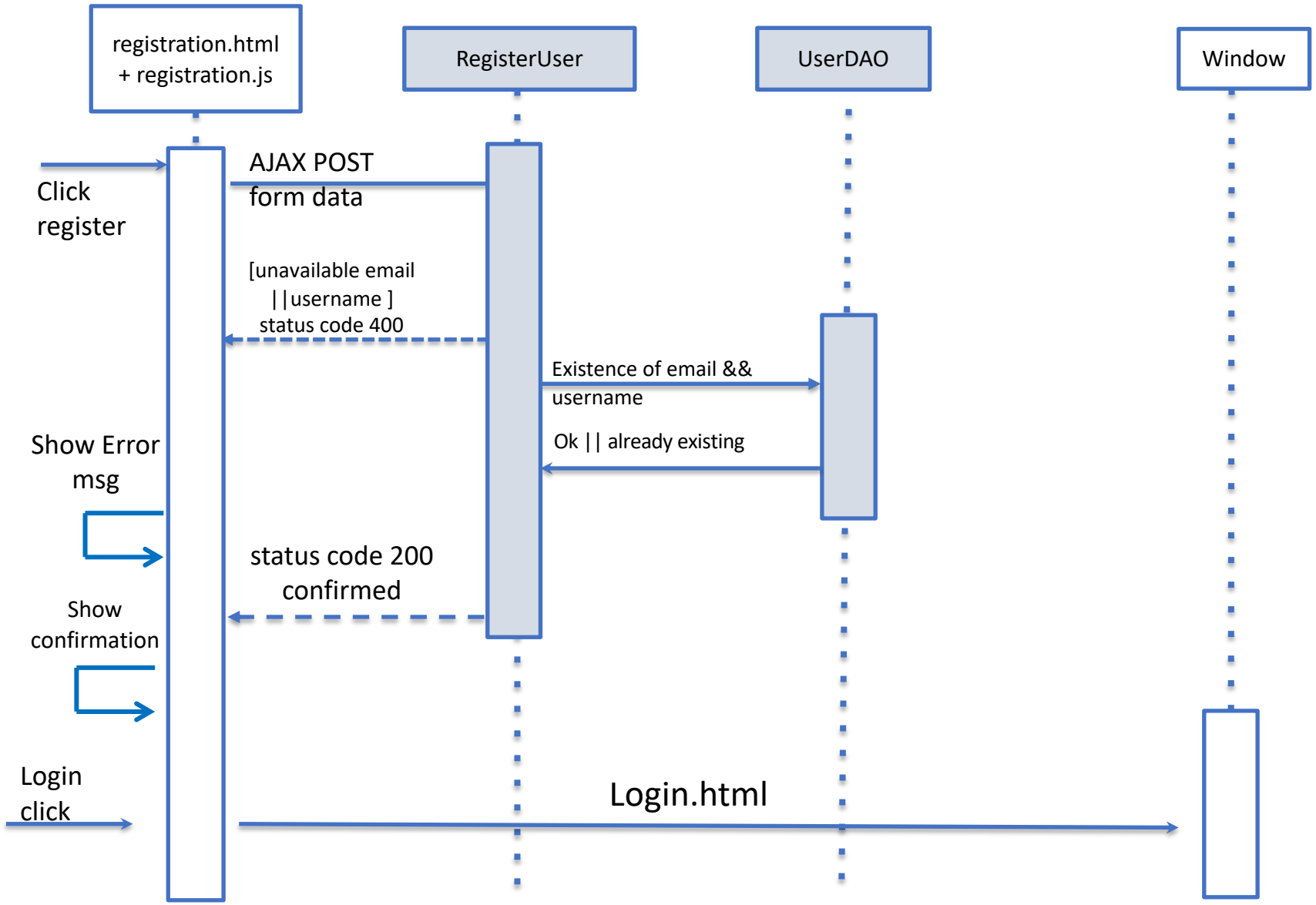
Controller / event handler

Client side		Server side	
Evento	**Controllore**	**Evento**	**Controllore**
index → login form → login	makeCall	POST (username password)	CheckLogin
Home page→ load	PageOrchestrator.start+ refresh	GET ()	GetAccountData+ GetAccountDetailsData
Home page→ accounts list→show	TransactionsList.update	GET (id account)	GetAccountDetailData
Home page→ accounts list→ click header	sort	-	-
Wizard → next, prev	changeStep	-	-
Wizard → cancel	reset	-	-
Wizard → submit	MakeCall	POST (transaction data)	CreateTransaction
Registration->registration form->register	MakeCall	POST(registration Data)	Register User

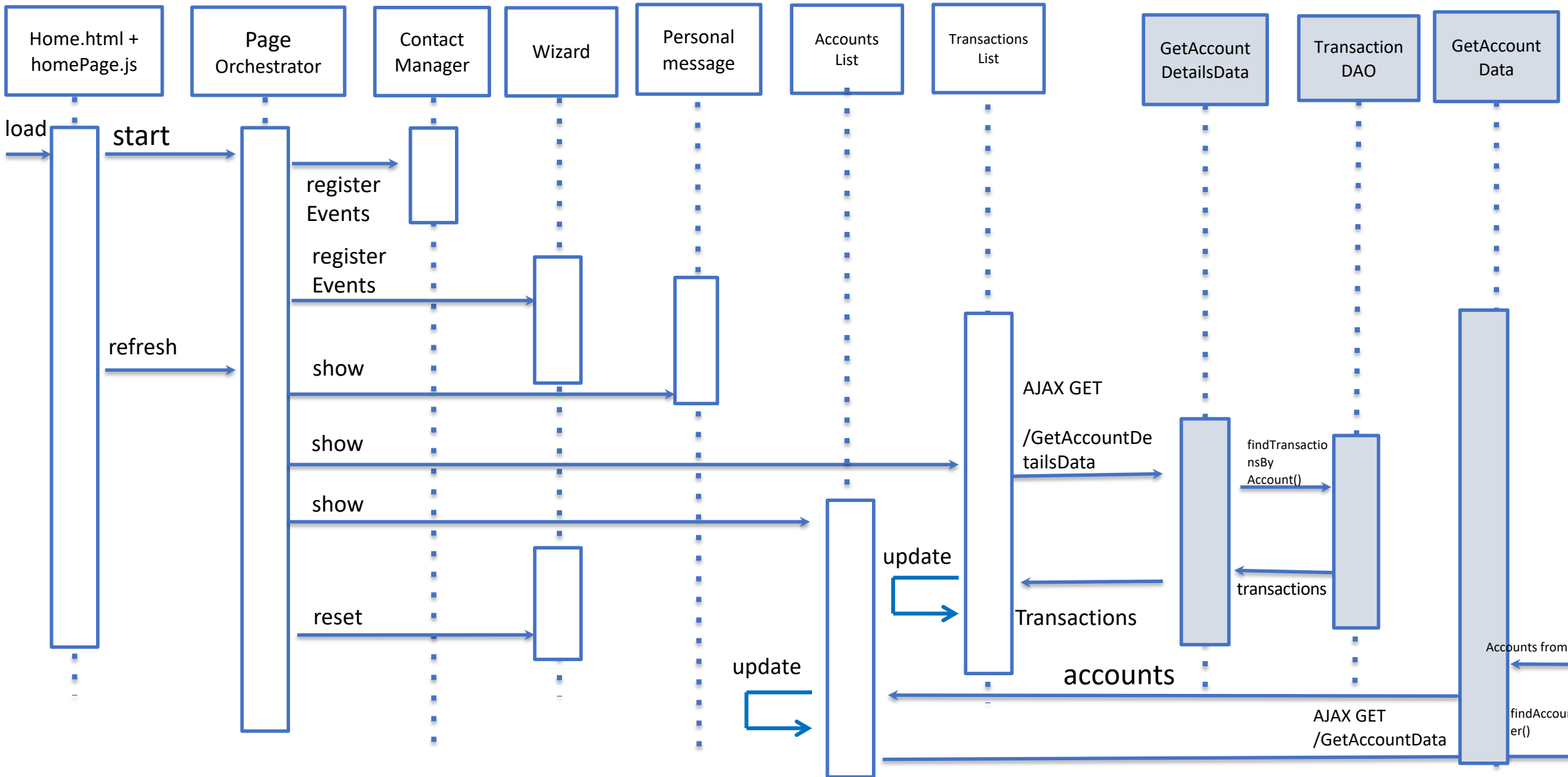
Event: login


Client side
Server side
index.html + checklogin.js
CheckLogin
UserDAO
Session
Session storage
Window
Click Login
AJAX POST
username
password
[username || pwd == null]
status code 400
checkCredentials
(user, pass)
user || null
Show error msg
[user == null]
status code 401
[user != null]
setAttribute ("user", user)
Set username
[user != null]
status code 200
username
[status code == 200]
[status code == 200]
[location = Home.html]

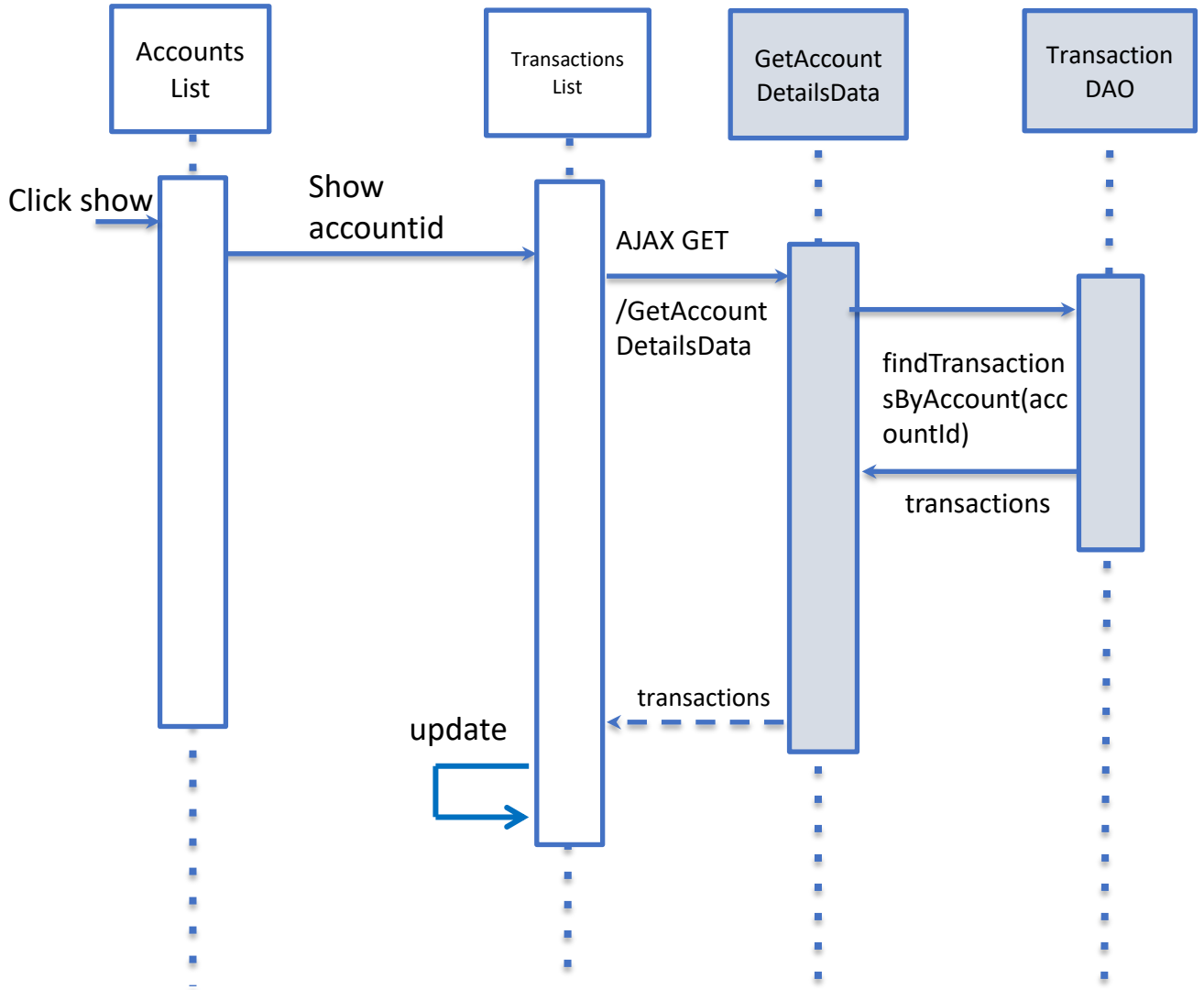
Event: registration



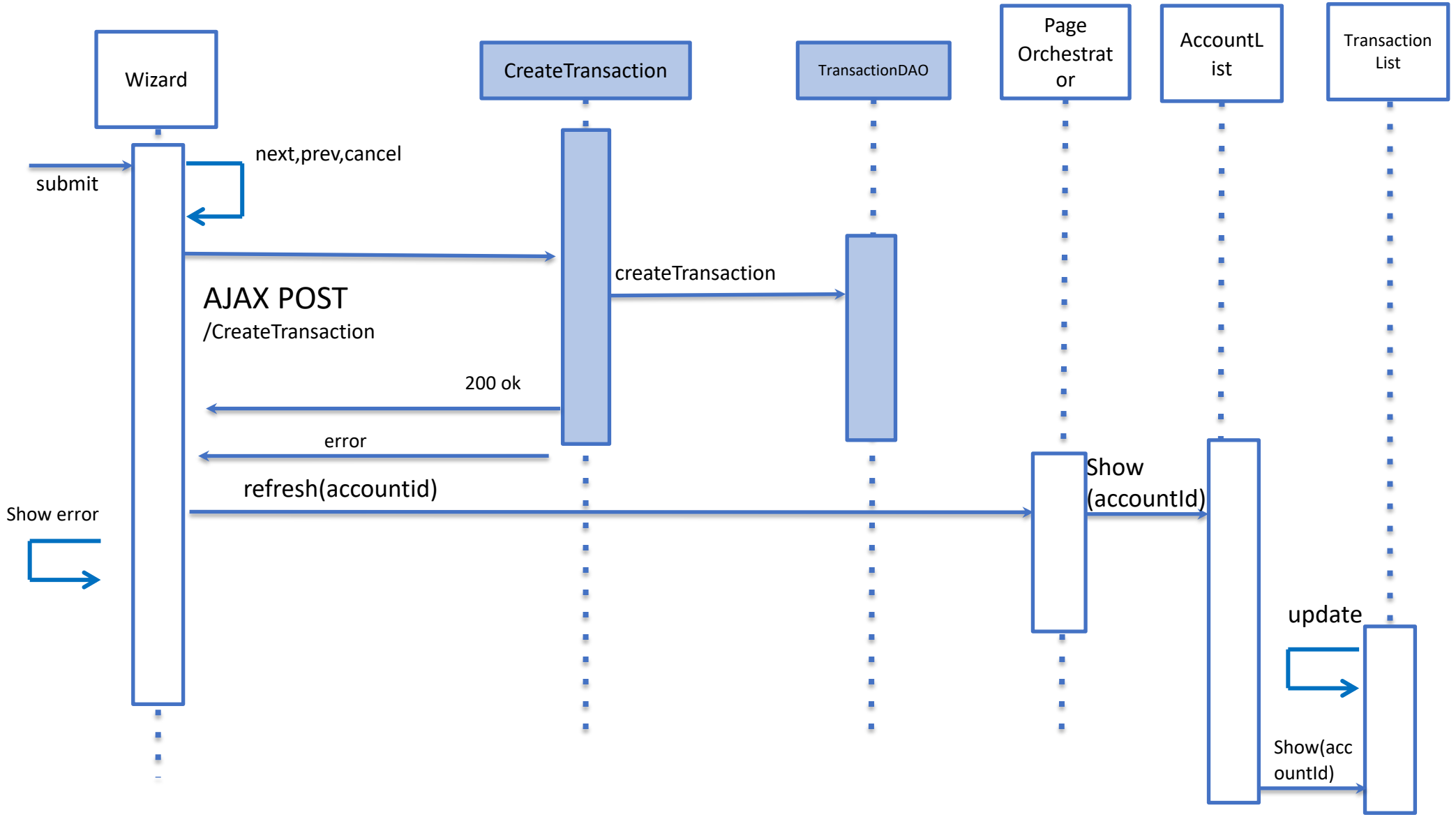
Event: loading Home page



Event: select an Account



Event: Create a Transaction



Evento: logout