Catalogazione di immagini

Tecnologie informatiche per il web – AA 2020/21

Esercizio 3 – Versione RIA

Leonardo Guerra – 10524955

Gaia Locchi – 10750598

Analisi dei dati (comuni alla versione HTML)

Un'applicazione permette all'**utente** (ad esempio il curatore di un catalogo online di immagini) di gestire una tassonomia di classificazione utile per etichettare immagini allo scopo di consentire la ricerca in base alla **categoria**. Dopo il login, l'utente accede a una pagina HOME in cui compare un albero gerarchico di categorie. Le categorie non dipendono dall'utente e sono in comune tra tutti gli utenti. Le categorie hanno **nomi** distinti. L'utente può inserire una nuova categoria nell'albero. Per fare ciò usa una form nella pagina HOME in cui specifica il nome della nuova categoria e sceglie la categoria padre. L'invio della nuova categoria comporta l'aggiornamento dell'albero: la nuova categoria **è appesa alla categoria padre** come ultimo sottoelemento. Alla nuova categoria viene assegnato un **codice numerico** che ne riflette la posizione. Per semplicità si ipotizzi che per ogni categoria il numero massimo di sottocategorie sia 9, numerate da 1 a 9. Dopo la creazione di una categoria, la pagina HOME mostra l'albero aggiornato. L'utente può spostare di posizione una categoria: per fare ciò clicca sul link "sposta" associato alla categoria da spostare. A seguito di tale azione l'applicazione mostra, sempre nella HOME page, l'albero con evidenziato il sotto albero attestato sulla categoria da spostare: tutte le altre categorie hanno un link "sposta qui". La selezione di un link "sposta qui" comporta l'inserimento della categoria da spostare come ultimo figlio della categoria destinazione. Le modifiche effettuate da un utente e salvate nella base di dati diventano visibili agli altri utenti. Lo spostamento potrebbe creare un vuoto nella numerazione delle categorie figlie dello stesso padre. La funzione di ricalcolo della numerazione delle categorie figlie dello stesso padre di quella spostata non è richiesta ma è lasciata come opzionale.

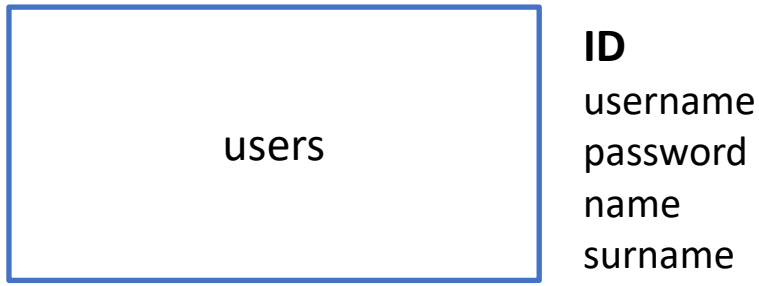
Entities, **attributes**, **relationships**

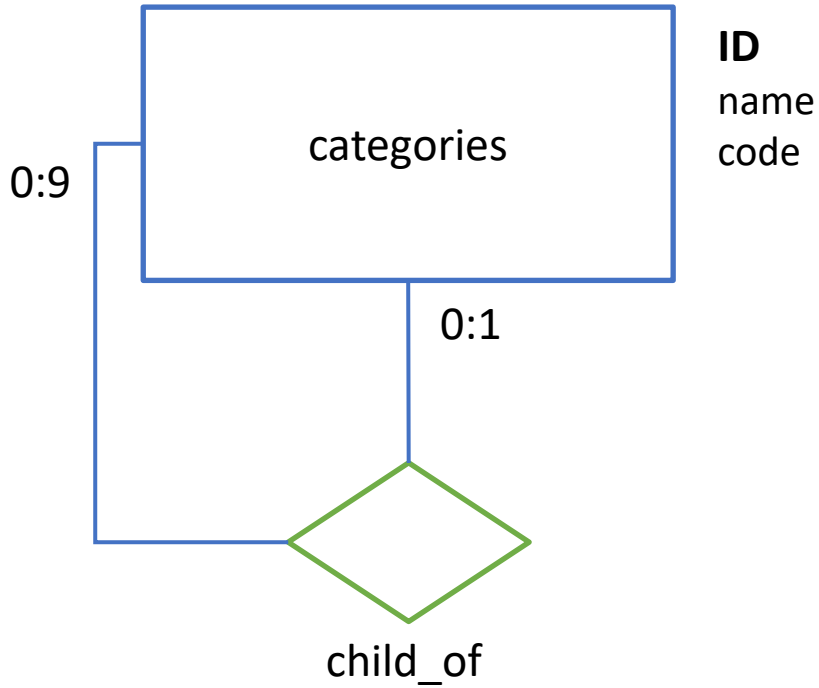
Specifiche RIA

Si realizzi un'applicazione client server web che estende e/o modifica le specifiche precedenti come segue:

- Dopo il login dell'utente, 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.
- La funzione di spostamento di una categoria è realizzata mediante drag & drop.
- A seguito del drop della categoria da spostare compare una finestra di dialogo con cui l'utente può confermare o cancellare lo spostamento. La conferma produce l'aggiornamento a lato client dell'albero.
- L'utente realizza spostamenti anche multipli a lato client. A seguito del primo spostamento compare un bottone SALVA la cui pressione provoca l'invio al server dell'elenco degli spostamenti realizzati (NON dell'intero albero). L'invio degli spostamenti produce l'aggiornamento dell'albero nella base dei dati e la comparsa di un messaggio di conferma dell'avvenuto salvataggio.

Database design





Local database schema

```
CREATE TABLE `users`
(
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(45) NOT NULL
UNIQUE,
`password` varchar(45) NOT NULL,
`name` varchar(45) NOT NULL,
`surname` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
)
```

```
CREATE TABLE `categories`
(
`id` int NOT NULL AUTO_INCREMENT,
`name` varchar(255) NOT NULL,
`code` varchar(255) NOT NULL,
`father` int,
PRIMARY KEY (`id`),
FOREIGN KEY (`father`) REFERENCES
`categories` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE
)
```

Analisi dei requisiti

Un'applicazione permette all'utente (ad esempio il curatore di un catalogo online di immagini) di gestire una tassonomia di classificazione utile per etichettare immagini allo scopo di consentire la ricerca in base alla categoria. Dopo il **login**, l'utente **accede a** una pagina **HOME** in cui compare **un albero gerarchico di categorie**. Le categorie non dipendono dall'utente e sono in comune tra tutti gli utenti. Le categorie hanno nomi distinti. L'utente può **inserire una nuova categoria** nell'albero. Per fare ciò usa una **form** nella pagina HOME in cui specifica il nome della nuova categoria e sceglie la categoria padre. L'**invio della nuova categoria** comporta l'**aggiornamento dell'albero**: la nuova categoria è appesa alla categoria padre come ultimo sottoelemento. Alla nuova categoria viene assegnato un codice numerico che ne riflette la posizione. Per semplicità si ipotizzi che per ogni categoria il numero massimo di sottocategorie sia 9, numerate da 1 a 9. Dopo la creazione di una categoria, la pagina HOME mostra l'albero aggiornato. L'utente può spostare di posizione una categoria: per fare ciò clicca sul link "sposta" associato alla categoria da spostare. A seguito di tale azione l'applicazione mostra, sempre nella HOME page, l'albero con evidenziato il sotto albero attestato sulla categoria da spostare: tutte le altre categorie hanno un link "sposta qui". La selezione di un link "sposta qui" comporta l'inserimento della categoria da spostare come ultimo figlio della categoria destinazione. Le modifiche effettuate da un utente e salvate nella base di dati diventano visibili agli altri utenti. Lo spostamento potrebbe creare un vuoto nella numerazione delle categorie figlie dello stesso padre. La funzione di ricalcolo della numerazione delle categorie figlie dello stesso padre di quella spostata non è richiesta ma è lasciata come opzionale.

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

Analisi dei requisiti (specifiche RIA)

Si realizzi un'applicazione client server web che estende e/o modifica le specifiche precedenti come segue:

- Dopo il login dell'utente, 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.
- La funzione di **spostamento di una categoria** è realizzata mediante **drag & drop**.
- A seguito del drop della categoria da spostare compare una **finestra di dialogo** con cui l'utente può **confermare o cancellare lo spostamento**. La conferma produce l'**aggiornamento a lato client** dell'albero.
- L'utente realizza spostamenti anche multipli a lato client. A seguito del primo spostamento compare un **bottone SALVA la cui pressione** provoca l'invio al server dell'elenco degli spostamenti realizzati (NON dell'intero albero). L'invio degli spostamenti produce l'**aggiornamento dell'albero** nella base dei dati e la comparsa di un **messaggio di conferma** dell'avvenuto salvataggio.

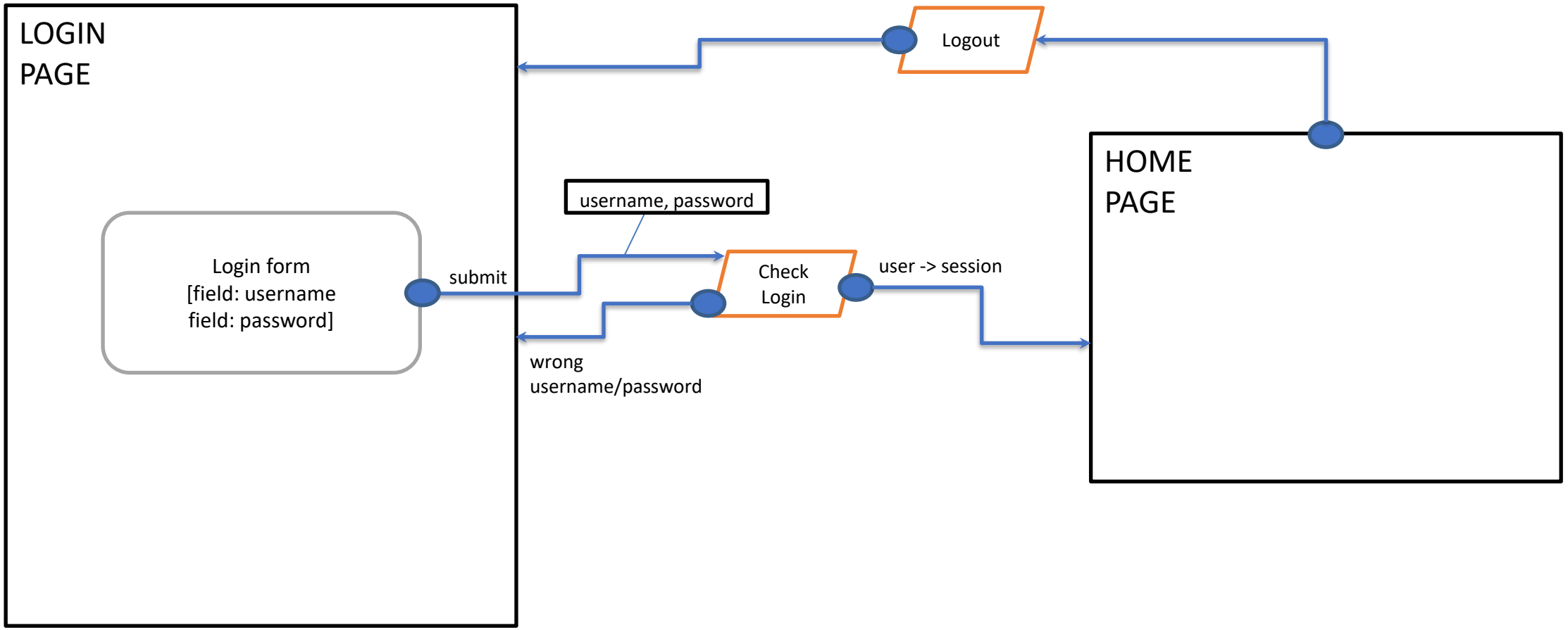
Pages (views), **view components**, **events**, **actions**

Completamento delle specifiche

- La **pagina di default** contiene la **form di login**.
- Nella form di login sia username che password sono obbligatori. Nella creazione di una nuova categoría, solo il nome della nuova categoría è obbligatorio.
- Le categorie “radice” (il cui codice è composto da una sola cifra) non hanno una categoria padre.
- **Dopo aver confermato il primo spostamento**, il **form di aggiunta di una nuova categoría è disabilitato** fino al salvataggio di tutte le modifiche effettuate.
- Al **refresh della home**, gli **spostamenti non salvati sono annullati**.
- Una categoría non può essere spostata come sotto-categoría dello stesso padre di partenza, nè di uno dei suoi figli (anche indiretti).
- **Funzione di ricalcolo**: se la categoría spostata è un “figlio intermedio” di una categoria (quindi ha “fratelli” con codice maggiore del suo), dopo il suo spostamento, per occupare il vuoto lasciato, il fratello con codice maggiore viene spostato (con tutto il suo albero) al suo posto, con conseguente aggiornamento di tutti i codici interessati.

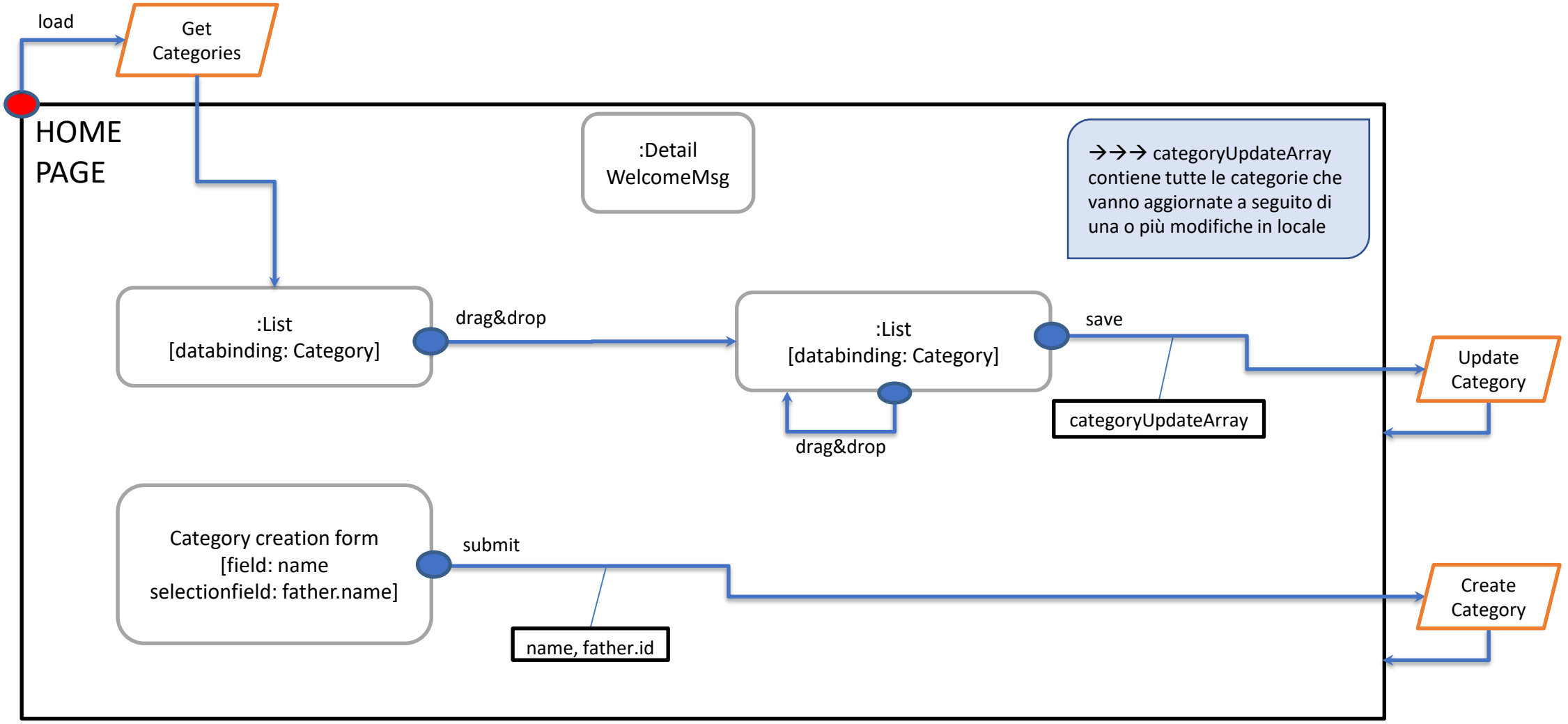
Pages (views), **view components**, **events**, **actions**

Application design



Application design

→→→ Dopo il primo salvataggio, il form nella home è disabilitato, finchè non viene premuto il tasto Salva



Eventi & azioni

→→→ 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 -> form Login -> submit	Controllo dati	POST (username, password)	Controllo credenziali
home -> load	Aggiornamento view con dati elenco	GET	Estrazione di tutte le categorie
home -> elenco categorie -> sposta categoria (drag&drop) -> Conferma/Cancella	Controllo dati (lato client) + Conferma: cambio posizione di una categoria (con eventuali relativi figli), Cancella: torna all'elenco precedente allo spostamento	-	-
home -> bottone Salva	-	POST (categoryUpdateArray)	Aggiorna l'elenco lato server
form Nuova categoria	Controllo dati	POST (name, fatherId)	Inserimento categoria
logout	-	POST	Terminazione della sessione

→→→ Dopo la conferma del primo spostamento, il form Nuova categoria è disabilitato fino al salvataggio

Controller / event handler

→→→ makeCall indica una funzione che fa una chiamata asincrona al server

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
index -> form Login -> submit	Funzione makeCall	POST (username, password)	CheckLogin (servlet)
home -> load	Funzione pageOrchestrator	GET	GetCategories (servlet)
home -> elenco categorie -> sposta categoria (drag&drop) -> Conferma/Cancella	Aggiorna view	-	-
home -> bottone Salva	Funzione makeCall	POST (categoryUpdateArray)	UpdateCategories (servlet)
form Nuova categoria	Funzione makeCall	POST (name, fatherId)	CreateCategory (servlet)
logout	-	POST	Logout (servlet)

Server side: DAO & model object

- Model objects (Beans)
  - User
  - Category
  - CategoryUpdate
- Controllers (Servlets)
  - CheckLogin
  - CreateCategory
  - GetCategories
  - UpdateCategories
  - Logout
- Data Access Objects (Classes)
  - UserDAO
    - checkCredentials(username, password)
  - CategoryDAO
    - createCategory(name, code, fatherId)
    - findLastChildCode(categoryId)
    - findCategoriesByFather(fatherId)
    - findAllCategories()
    - getNumOfRoots()
    - getCategorySubtree(categoryId)
    - findCategoryCode(categoryId)
    - findMaxRootCode()
    - updateCategories(categoryUpdateArray)
    - existsCategory(name)

Client side: view & view component

- index
  - Login form
    - gestione della submit e degli errori
- home
  - CategoriesList
    - reset(): imposta le condizioni di iniziali visibilità dei componenti
    - show(): richiede al server i dati delle categorie
    - registerEvents(): associa al componente le funzioni per gestirne gli eventi
    - update(): riceve i dati dal server e aggiorna la lista delle categorie
  - CategoryForm
    - reset(): imposta le condizioni di iniziali visibilità
    - disable(): disabilita gli input del form
    - enable(): abilita gli input del form
    - registerEvents(): associa al componente le funzioni per gestirne gli eventi
    - show(): richiede al server i dati delle categorie
    - update(): riceve i dati dal server e aggiorna la lista dei possibili padri
  - UpdateModal
    - show(): imposta le condizioni di visibilità successive al drop
    - close(): imposta le condizioni di visibilità successive all'annullamento dello spostamento
    - confirm(): sposta la nuova categoria successivamente a un drop legittimo
    - reset(): imposta le condizioni di iniziali visibilità
  - AfterUpdateModal
    - show(), close(), reset(): [come sopra]
  - PersonalMessage
    - show(): mostra nome e cognome dell'utente corrente, in alto a sinistra nella pagina

Gestione del ciclo di vita

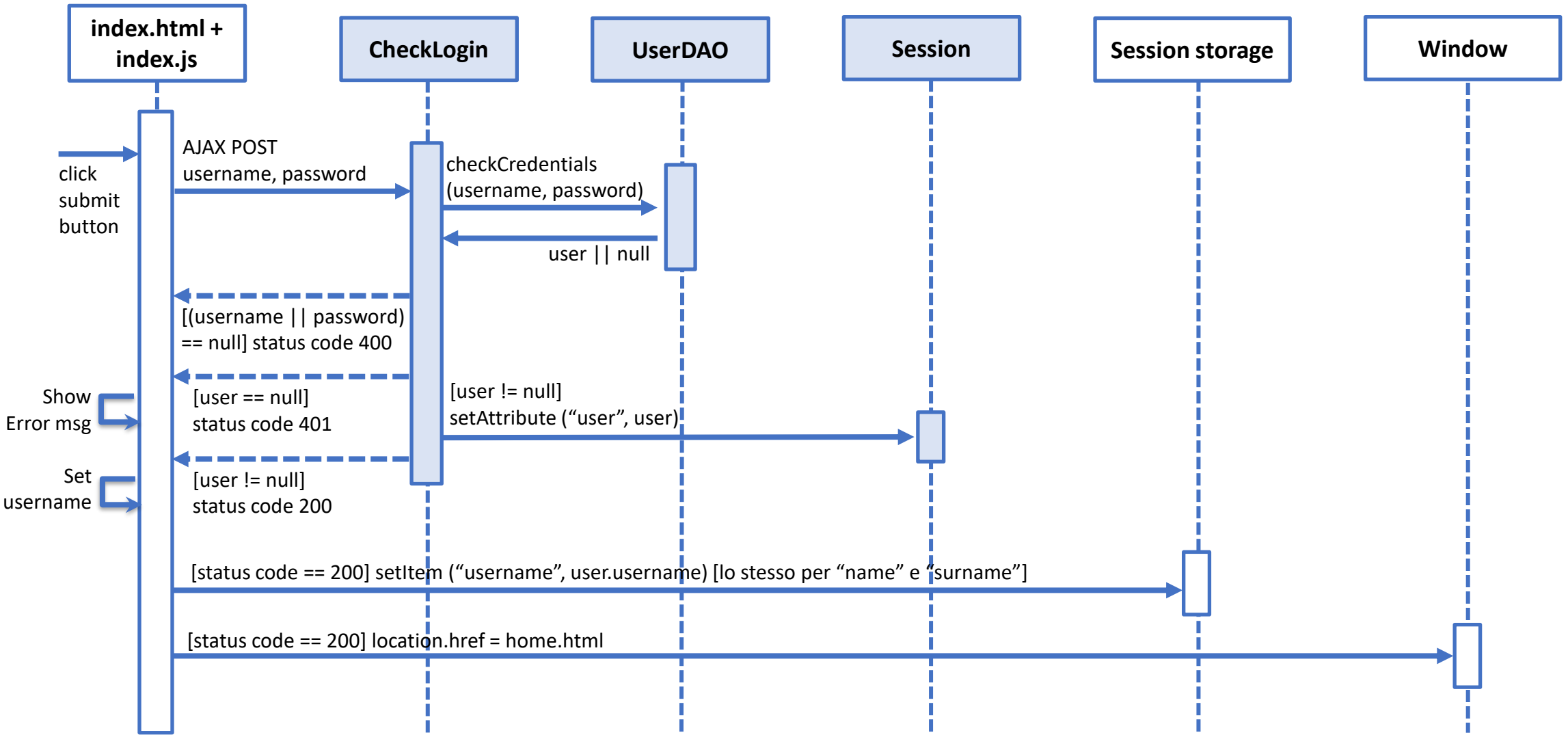
- PageOrchestrator
  - start(): crea e inizializza il personal message, la lista delle categorie, la form per la nuova categoria e le modali per la richiesta di conferma dell'update in locale e per l'avviso di salvataggio su server avvenuto correttamente
  - refresh(): resetta e mostra la lista delle categorie e il form per la creazione della nuova categoria, e resetta le modali per la richiesta di conferma dell'update in locale e per l'avviso di salvataggio su server avvenuto correttamente (ma non il personal message)

Event: login

Legenda

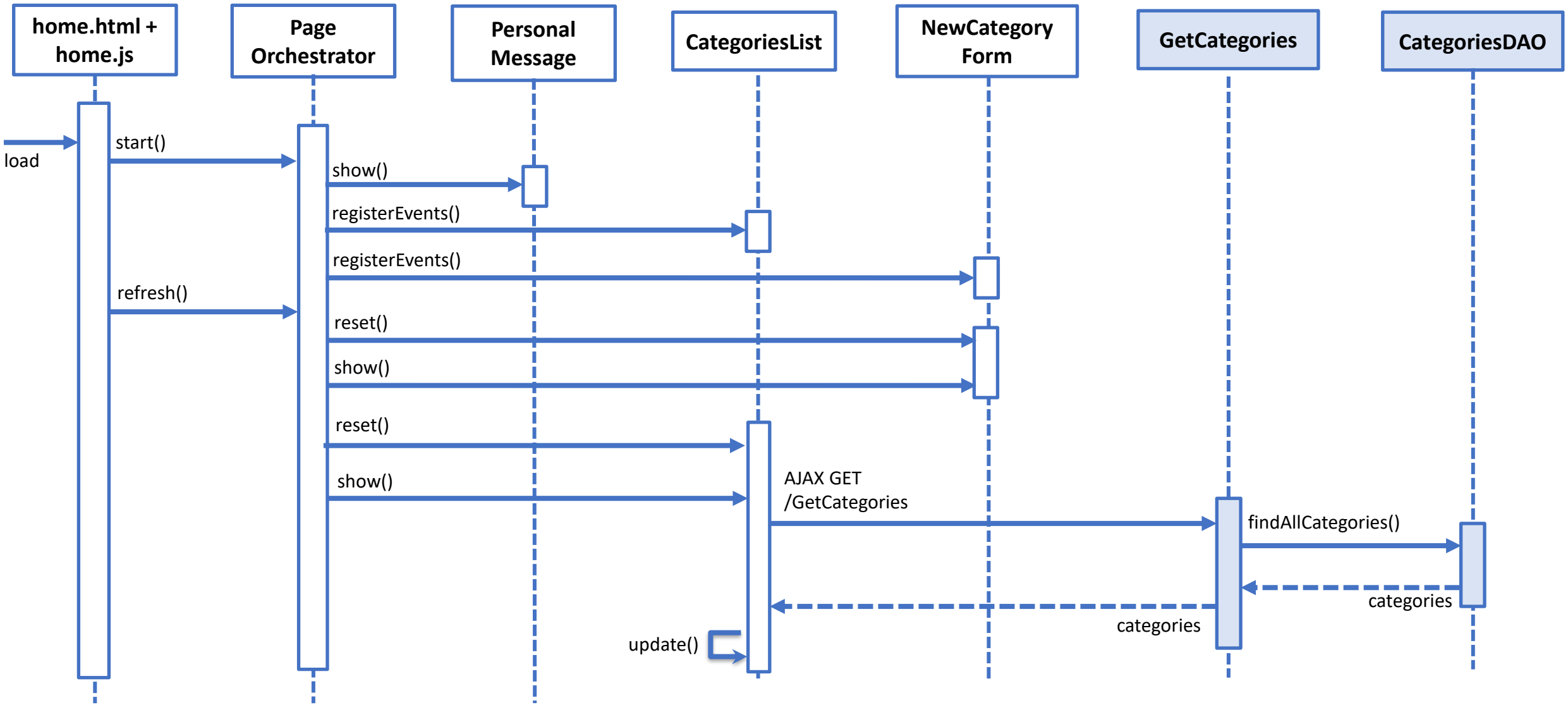
Client Side | *Server side*

→→→ Per brevità, nei prossimi diagrammi la gestione di quasi tutti gli errori verrà omessa

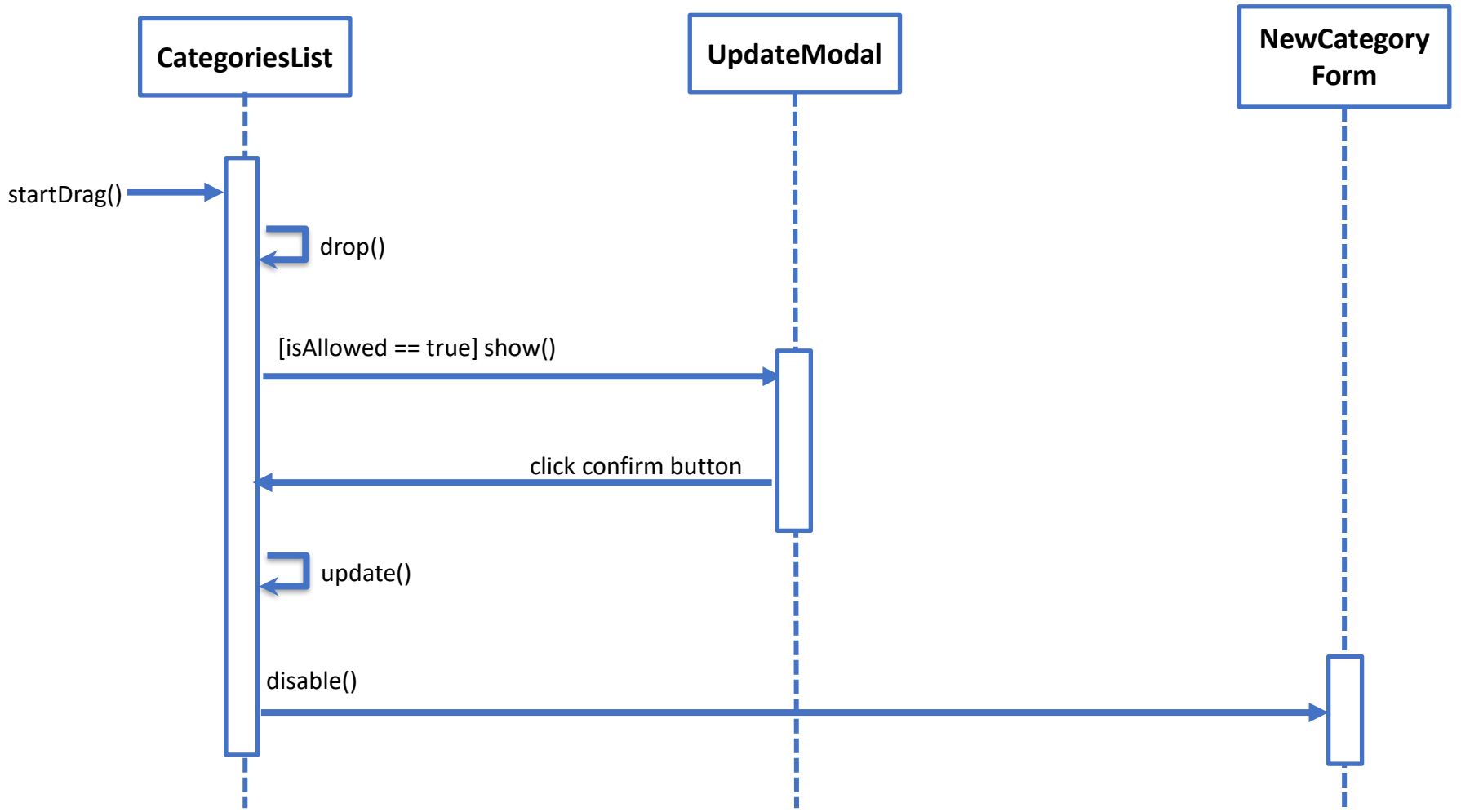


Event: caricamento home

→→→ Anche UpdateModal e AfterUpdateModal vengono resettate al refresh (non inserite per questioni di leggibilità dello schema)

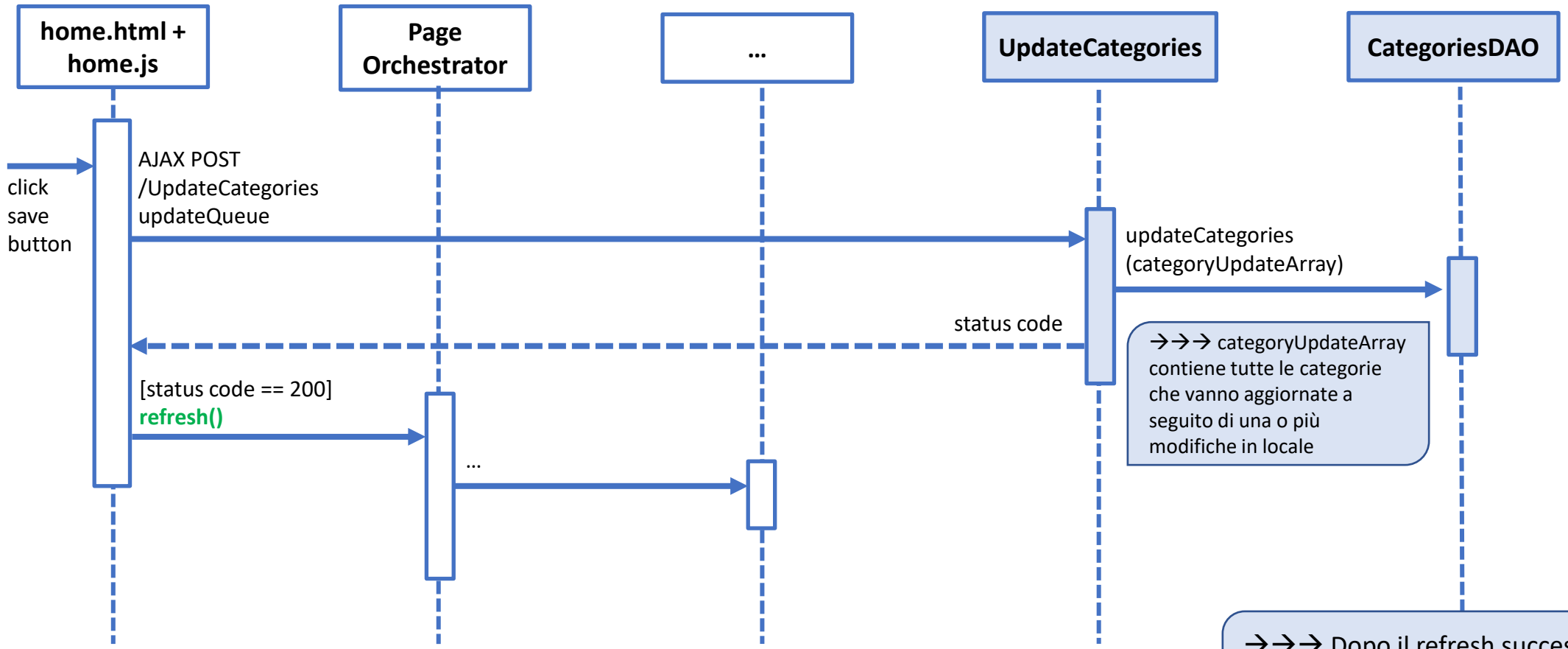


Event: spostamento categorie



Event: salvataggio spostamenti

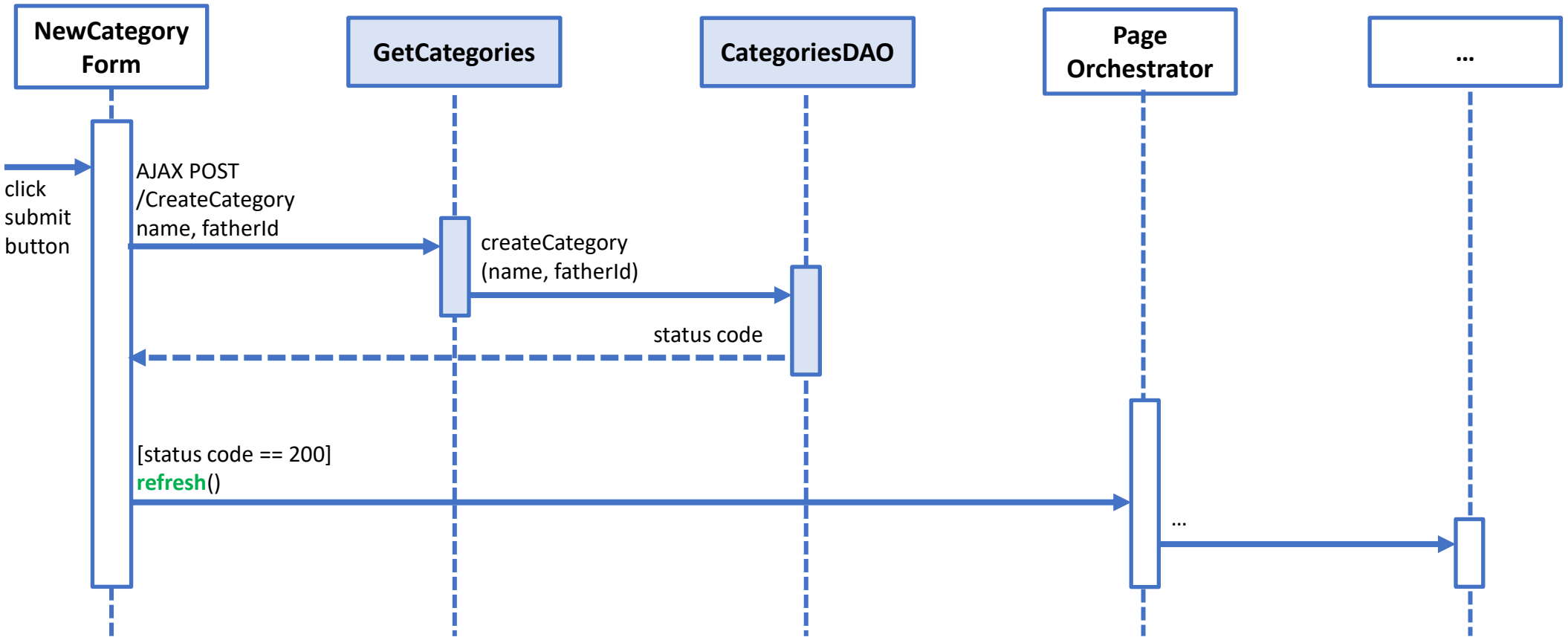
→→→ Le interazioni successive al **refresh** della home (indicate da «…») non sono riportate per brevità, ma sono presenti nella slide «Event: caricamento home»



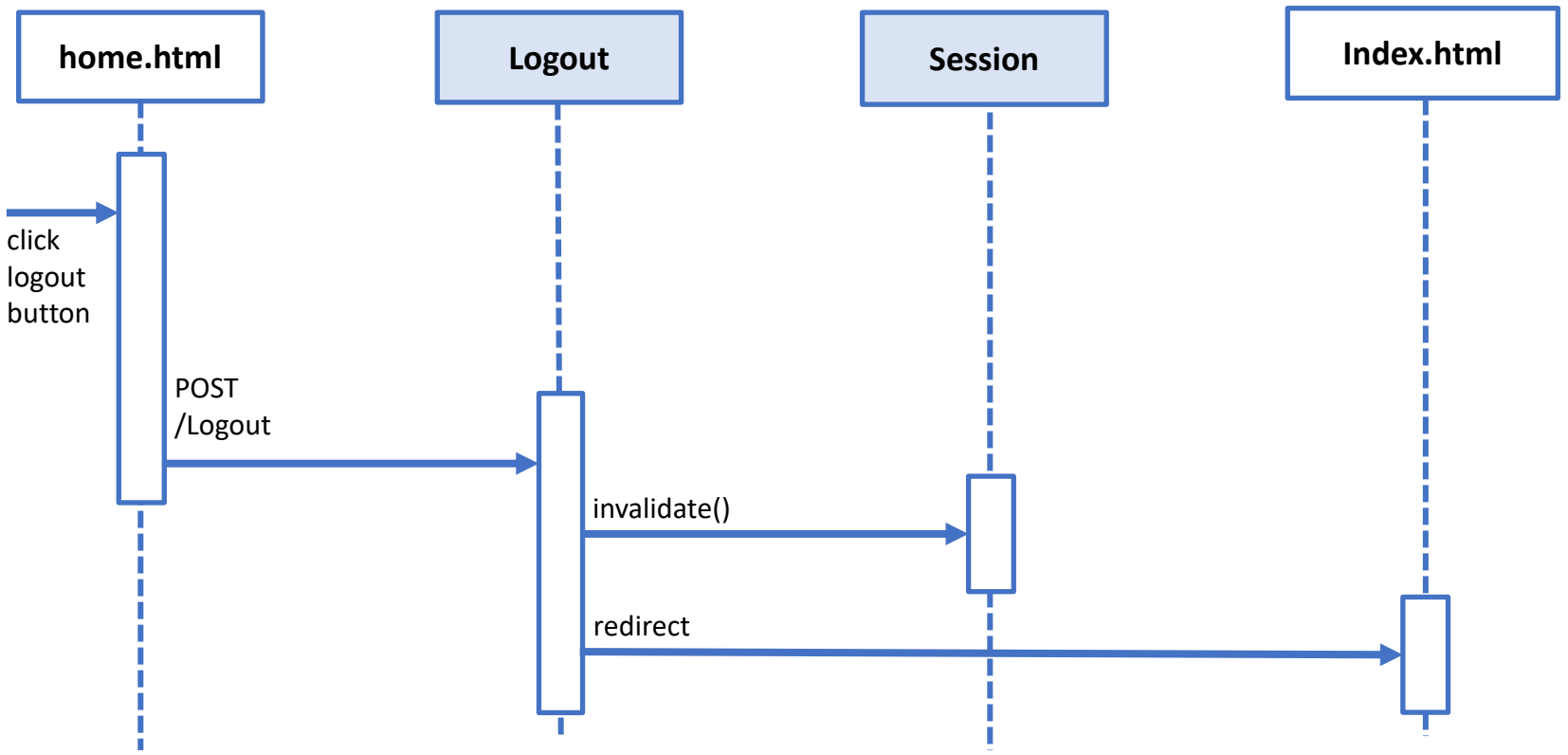
→→→ Dopo il refresh successivo al salvataggio, viene invocato il metodo show() di AfterUpdateModal

Event: aggiunta categoria

→→→ Le interazioni successive al **refresh** della home (indicate da «…») non sono riportate per brevità, ma sono presenti nella slide «Event: caricamento home»



Event: logout



Packages Utils e Filters

- Package Utils
  - ConnectionHandler
    - Si occupa di istanziare e chiudere la Connection
- Package Filters
  - LoginChecker
    - Classe utilizzata ogni volta in cui un utente realizza un’azione nella pagina web (GET/POST), che si occupa di controllare la validità della sessione