Progetto di tecnologie informatiche per il web

Carrello con più fornitori e ordine

Anno 2022-2023

A cura di Lucia Famoso

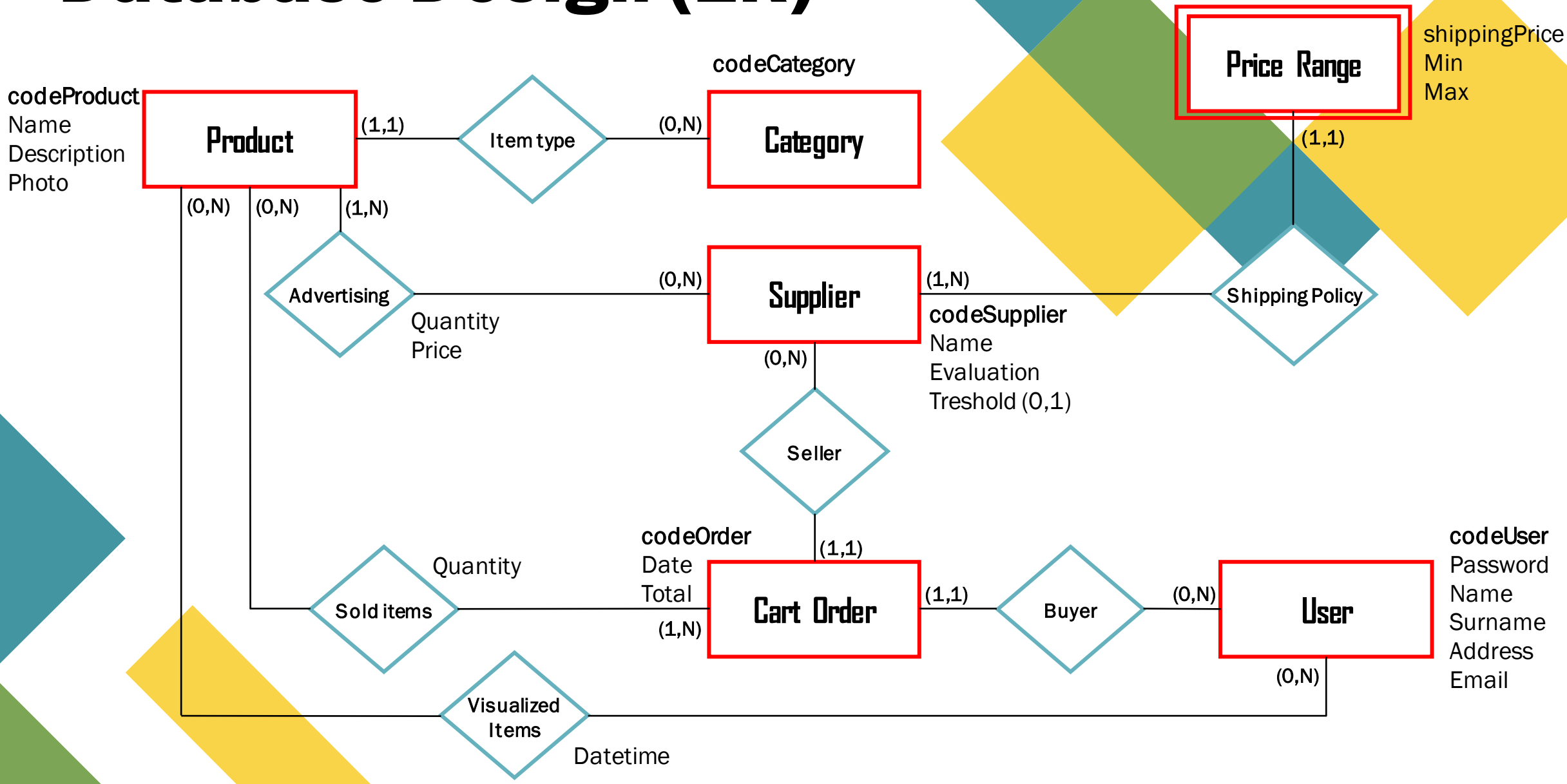
Analisi traccia

Un’applicazione di commercio elettronico consente all’utente (acquirente) di visualizzare un catalogo di prodotti venduti da diversi fornitori, inserire prodotti in un carrello della spesa e creare un ordine di acquisto a partire dal contenuto del carrello.
Un **prodotto** ha un **codice** (campo chiave), un **nome**, una **descrizione**, una **categoria** merceologica e una **foto**. Lo stesso **prodotto (cioè codice prodotto) può essere venduto da più fornitori** a prezzi differenti.
Un **fornitore** ha un **codice**, un **nome**, una **valutazione** da 1 a 5 stelle e una **politica di spedizione**.
Un **utente** ha un nome, un cognome, un’e-mail, una password e un indirizzo di spedizione.
La politica di spedizione precisa il prezzo della spedizione in base al numero di articoli ordinati.
Ogni fornitore è libero di definire fasce di spesa.
Una **fascia di spesa** ha un **numero minimo**, un **numero massimo** e un **prezzo**. Ad esempio: da 1 a 3 articoli 15€, da 4 a 10 articoli 20€, oltre a 10 articoli, ecc. Oltre alla fascia di spesa, il fornitore può anche indicare un **importo in euro oltre al quale la spedizione è gratuita**. Se il totale supera la soglia per la gratuità della spedizione, la spedizione è gratuita indipendentemente dal numero di articoli.

Dopo il **login**, l'utente accede a una pagina HOME che mostra (come tutte le altre pagine) un menù con i link **HOME**, **CARRELLO**, **ORDINI**, un campo di ricerca e una lista degli ultimi cinque prodotti visualizzati dall'utente. Se l'utente non ha visualizzato almeno cinque prodotti, la lista è completata con prodotti in offerta scelti a caso in una categoria di default.
L'utente può inserire una parola chiave di ricerca nel campo di input e premere INVIO. A seguito dell'invio compare una pagina **RISULTATI** con (una lista di) prodotti che contengono la chiave di ricerca nel nome o nella descrizione. L'elenco mostra solo il codice, il nome del prodotto e il prezzo minimo di vendita del prodotto da parte dei fornitori che lo vendono (lo stesso prodotto può essere venduto da diversi fornitori a prezzi diversi e l'elenco mostra il minimo valore di tali prezzi). L'elenco è ordinato in modo crescente in base al prezzo minimo di vendita del prodotto da parte dei fornitori che lo offrono. L'utente può selezionare mediante un click un elemento dell'elenco e visualizzare nella stessa pagina i dati completi e l'elenco dei fornitori che lo vendono a vari prezzi (questa azione rende il prodotto "**visualizzato**"). Per ogni fornitore in tale elenco compaiono: nome, valutazione, prezzo unitario, fasce di spesa di spedizione, importo minimo della spedizione gratuita e il numero dei prodotti e valore totale dei prodotti di quel fornitore che l'utente ha già messo nel carrello.

Accanto all'offerta di ciascun fornitore compare un campo di input intero (quantità) e un bottone METTI NEL CARRELLO. L'inserimento nel carrello di una quantità maggiore di zero di prodotti comporta l'aggiornamento del contenuto del carrello e la visualizzazione della pagina CARRELLO. Questa mostra i prodotti inseriti, raggruppati per fornitore.
Per ogni fornitore nel carrello si vedono la lista dei prodotti, il prezzo totale dei prodotti e il prezzo della spedizione calcolato in base alla politica del fornitore. Per ogni fornitore compare un bottone ORDINA.
Premere il bottone comporta l'eliminazione dei prodotti del fornitore dal carrello e la creazione di un ordine corrispondente.
Un **ordine** ha un codice, il nome del fornitore, l'elenco dei prodotti, un valore totale composto dalla somma del valore dei prodotti e delle spese di spedizione, una data di spedizione e l'indirizzo di spedizione dell'utente. I valori degli attributi di un ordine sono memorizzati esplicitamente nella base di dati indipendentemente dai dati del carrello. In ogni momento l'utente può accedere tramite il menu alle pagine HOME, ORDINI e CARRELLO. La pagina ORDINI mostra l'elenco ordinato per data decrescente degli ordini con tutti i dati associati.

Database Design (ER)



Create Statements

```
CREATE TABLE `product` (
  `codeProduct` int NOT NULL AUTO_INCREMENT,
  `name` varchar(64) NOT NULL,
  `description` varchar(256) NOT NULL,
  `codeCategory` varchar(45) NOT NULL,
  `photo` longblob,

  PRIMARY KEY (`codeProduct`),
  KEY `category_idx` (`codeCategory`),
  CONSTRAINT `category-prod` FOREIGN KEY
(`codeCategory`) REFERENCES `category`
(`codeCategory`) ON UPDATE CASCADE
)
```

```
CREATE TABLE `category` (
  `codeCategory` varchar(45) NOT NULL,
  PRIMARY KEY (`codeCategory`)
)
```

```
CREATE TABLE `supplier` (
  `codeSupplier` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `evaluation` decimal(2,1) unsigned NOT NULL DEFAULT '0.0',
  `treshold` decimal(7,2) DEFAULT NULL,
  PRIMARY KEY (`codeSupplier`),
  CONSTRAINT `supplier_chk_1` CHECK ((`evaluation`
between 0.0 and 5.0))
)
```

```
CREATE TABLE `price_range` (
  `codeSupplier` int NOT NULL,
  `min` int NOT NULL,
  `max` int NOT NULL,
  `shippingPrice` decimal(7,2) NOT NULL,
  PRIMARY KEY (`codeSupplier`,`min`),
  KEY `supplier-range_idx` (`codeSupplier`),
  CONSTRAINT `supplier-range` FOREIGN KEY
(`codeSupplier`) REFERENCES `supplier`
(`codeSupplier`) ON DELETE CASCADE ON UPDATE
CASCADE
)
```

```
CREATE TABLE `user` (
  `codeUser` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `name` varchar(125) NOT NULL,
  `surname` varchar(125) NOT NULL,
  `address` varchar(256) NOT NULL,
  `email` varchar(45) NOT NULL,
  PRIMARY KEY (`codeUser`),
  UNIQUE KEY `email_UNIQUE` (`email`)
)
CREATE TABLE `visualized_items` (
  `codeUser` varchar(45) NOT NULL,
  `codeProduct` int NOT NULL,
  `dateTime` datetime NOT NULL,
  PRIMARY KEY (`codeUser`,`codeProduct`),
  KEY `prod-visualized_idx` (`codeProduct`),
  CONSTRAINT `prod-visualized` FOREIGN KEY
(`codeProduct`) REFERENCES `product`
(`codeProduct`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `user-visualized` FOREIGN KEY
(`codeUser`) REFERENCES `user` (`codeUser`) ON
DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `advertising` (
  `codeProduct` int NOT NULL,
  `codeSupplier` int NOT NULL,
  `quantity` int NOT NULL DEFAULT '0',
  `price` decimal(7,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (`codeProduct`,`codeSupplier`),
  KEY `codeSupplier_idx` (`codeSupplier`),
  CONSTRAINT `product-adv` FOREIGN KEY
(`codeProduct`) REFERENCES `product`
(`codeProduct`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `supplier-adv` FOREIGN KEY
(`codeSupplier`) REFERENCES `supplier`
(`codeSupplier`) ON DELETE CASCADE ON UPDATE
CASCADE
)
```

```
CREATE TABLE `cart_order` (
  `codeOrder` int NOT NULL AUTO_INCREMENT,
  `codeSupplier` int NOT NULL,
  `codeUser` varchar(45) NOT NULL,
  `date` datetime NOT NULL,
  `total` decimal(7,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (`codeOrder`),
  KEY `supplier-order_idx` (`codeSupplier`),
  KEY `user-order_idx` (`codeUser`),
  CONSTRAINT `supplier-order` FOREIGN KEY
(`codeSupplier`) REFERENCES `supplier`
(`codeSupplier`) ON UPDATE CASCADE,
  CONSTRAINT `user-order` FOREIGN KEY (`codeUser`)
REFERENCES `user` (`codeUser`) ON UPDATE CASCADE
)
```

```
CREATE TABLE `sold_items` (
  `codeOrder` int NOT NULL,
  `codeProduct` int NOT NULL,
  `quantity` int NOT NULL,
  PRIMARY KEY (`codeOrder`,`codeProduct`),
  KEY `product-item_idx` (`codeProduct`),
  KEY `order-sold_idx` (`codeOrder`),
  CONSTRAINT `order-sold` FOREIGN KEY (`codeOrder`)
REFERENCES `cart_order` (`codeOrder`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `product-sold` FOREIGN KEY
(`codeProduct`) REFERENCES `product`
(`codeProduct`) ON UPDATE CASCADE
)
```

Versione HTML pure

Application Design (IFML)

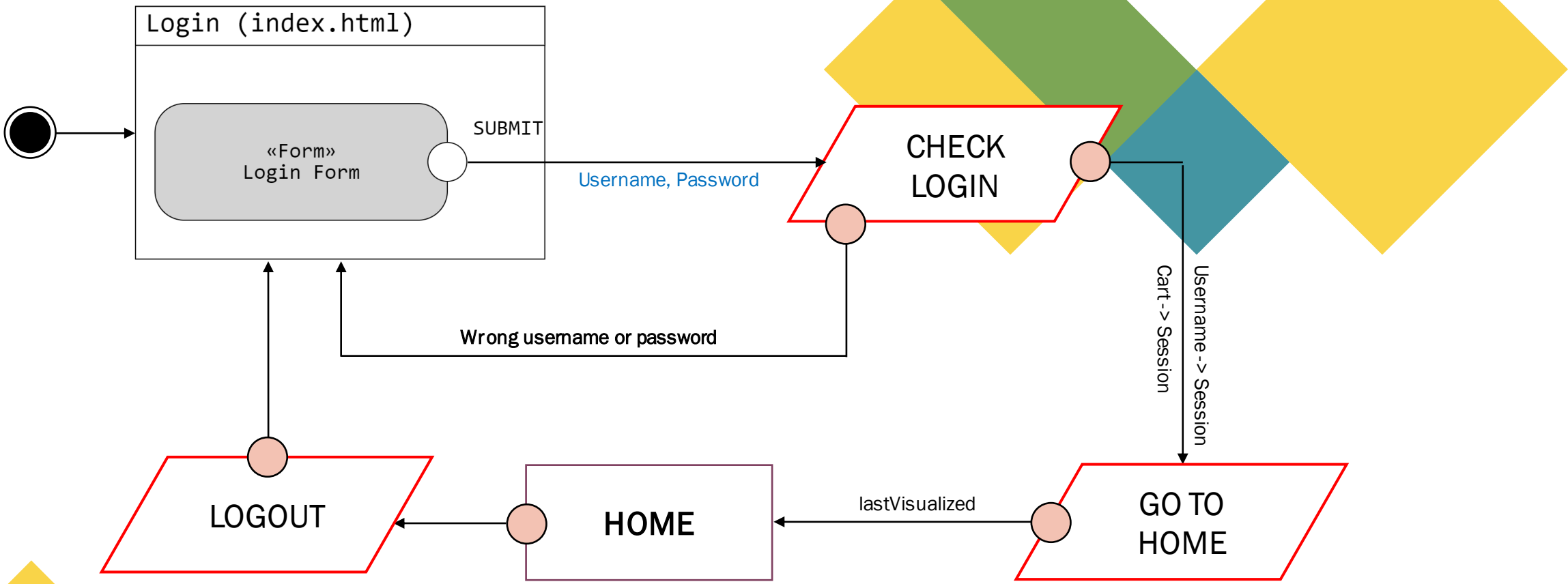
Sono state eseguite alcune esemplificazioni allo schema al fine di evitare difficoltà nella lettura:

- Le interazioni sono state spezzate in più diagrammi IFML per agevolare la comprensione
- I cambi di vista attuabili tramite le interazioni con il menù sono stati omessi, eccetto per la home page, in quanto ripetuti per ognuna delle pagine. Tali interazioni comprendono il **logout**.
- Alcune chiamate alle servlet adibite puramente al rendering (**GoToHome**/**GoToCart**) sono state omesse per brevità e per portare in primo piano le servlet contenenti la logica computazionale (**AddToCart**/**RemoveFromCart**).

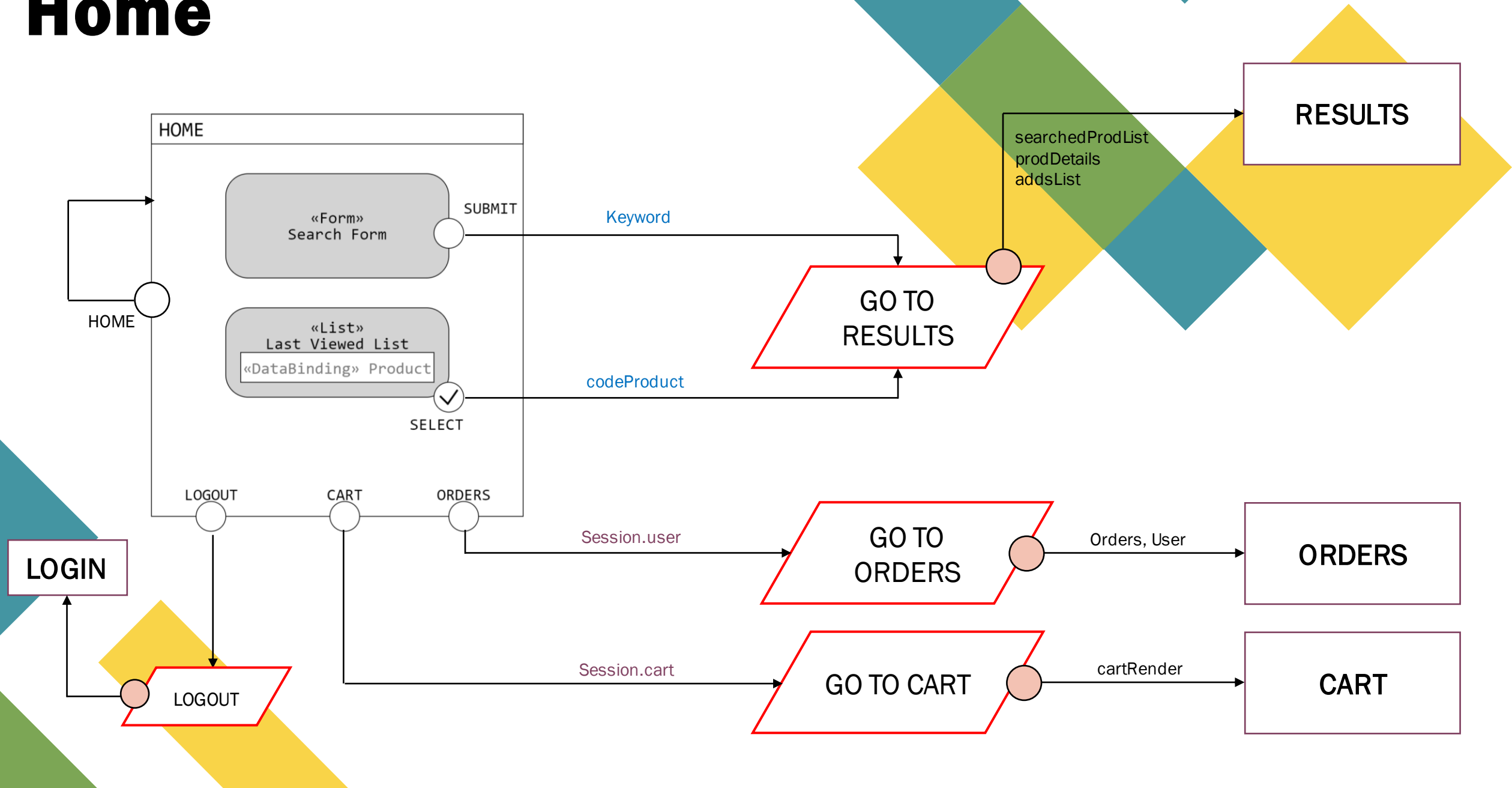
  E' stata messa in atto una differenziazione tra servlet-logiche e servlet-render al fine di ridurre al minimo, dove possibile, le incongruenze tra i dati nella sessione e i dati all'interno del database.

  L'esempio principale di questa scelta implementativa lo si può ritrovare nella gestione del carrello:
  Il carrello all'interno della sessione, difatti, contiene informazioni minime riguardanti gli elementi al suo interno: chiavi primarie e quantità.
  Dati come il prezzo di vendita e i dettagli del prodotto vengono di conseguenza caricati direttamente dal database e controllati ad ogni chiamata della servlet di rendering: **GoToCart**, al fine di mantenere la totale congruenza con i dati all'interno del database.

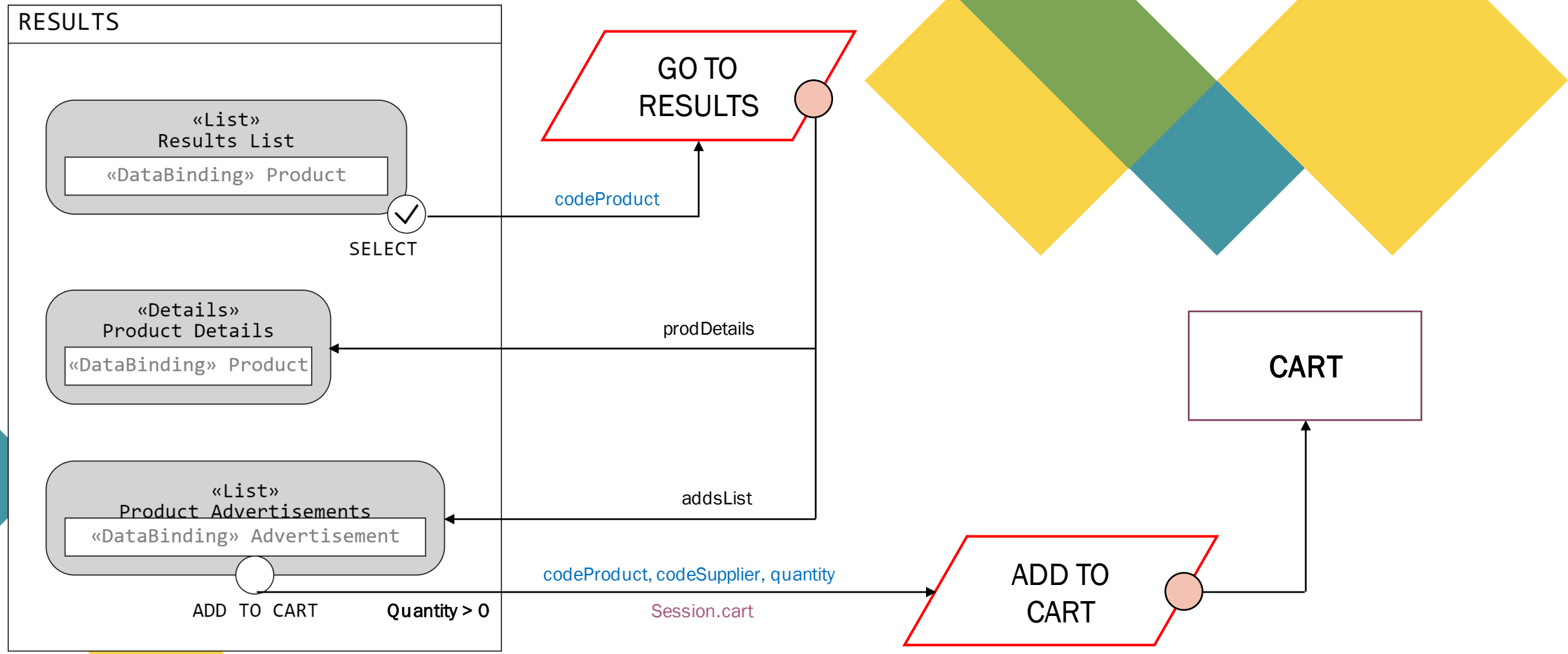
Login


Login (index.html)
«Form»
Login Form
SUBMIT
Username, Password
CHECK
LOGIN
Wrong username or password
Cart-> Session
Username -> Session
LOGOUT
HOME
lastVisualized
GO TO
HOME

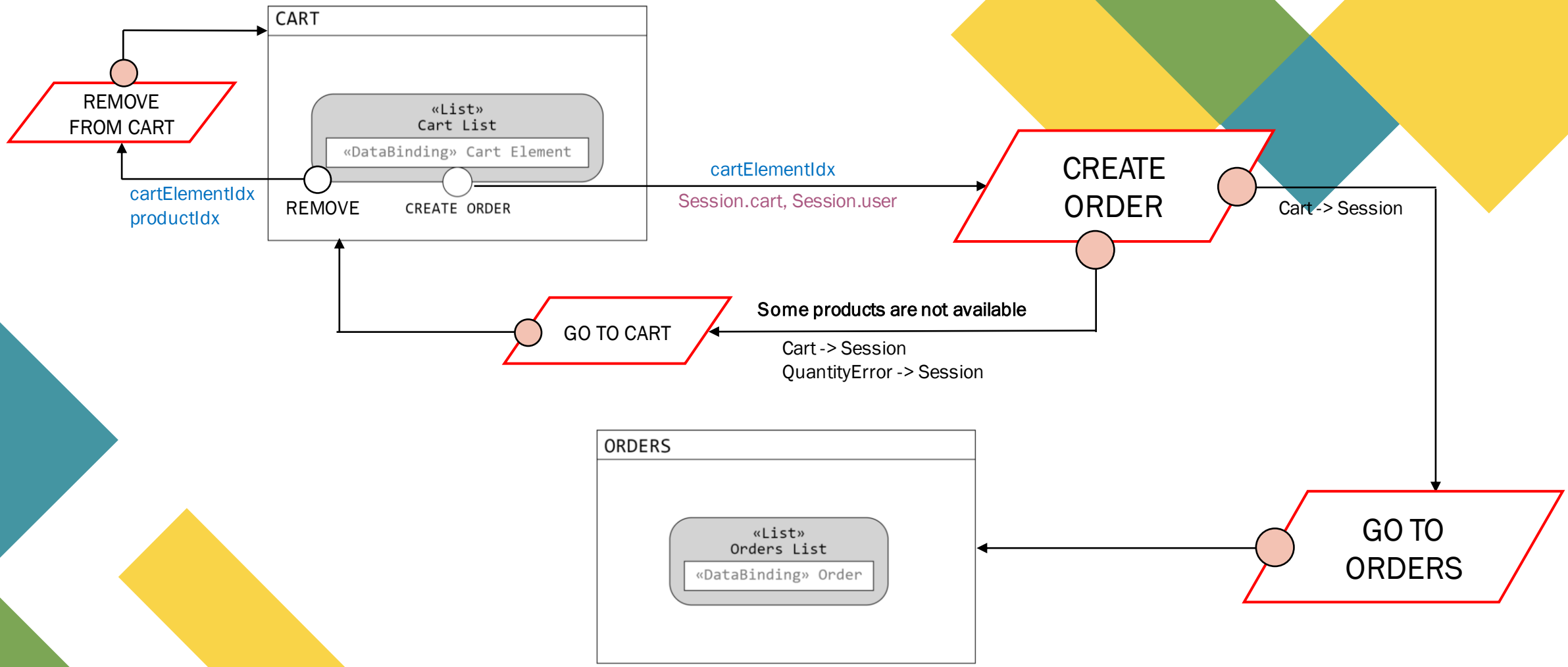
Home


HOME
«Form»
Search Form
SUBMIT
HOME
«List»
Last Viewed List
«DataBinding» Product
SELECT
LOGOUT
CART
ORDERS
Keyword
codeProduct
searchedProdList
prodDetails
addsList
RESULTS
GO TO
RESULTS
Session.user
GO TO
ORDERS
Orders, User
ORDERS
LOGIN
LOGOUT
Session.cart
GO TO CART
cartRender
CART

Results


RESULTS
«List»
Results List
«DataBinding» Product
SELECT
«Details»
Product Details
«DataBinding» Product
«List»
Product Advertisements
«DataBinding» Advertisement
ADD TO CART
Quantity > 0
GO TO RESULTS
codeProduct
prodDetails
addsList
codeProduct, codeSupplier, quantity
Session.cart
ADD TO CART
CART

Cart and Orders



Components

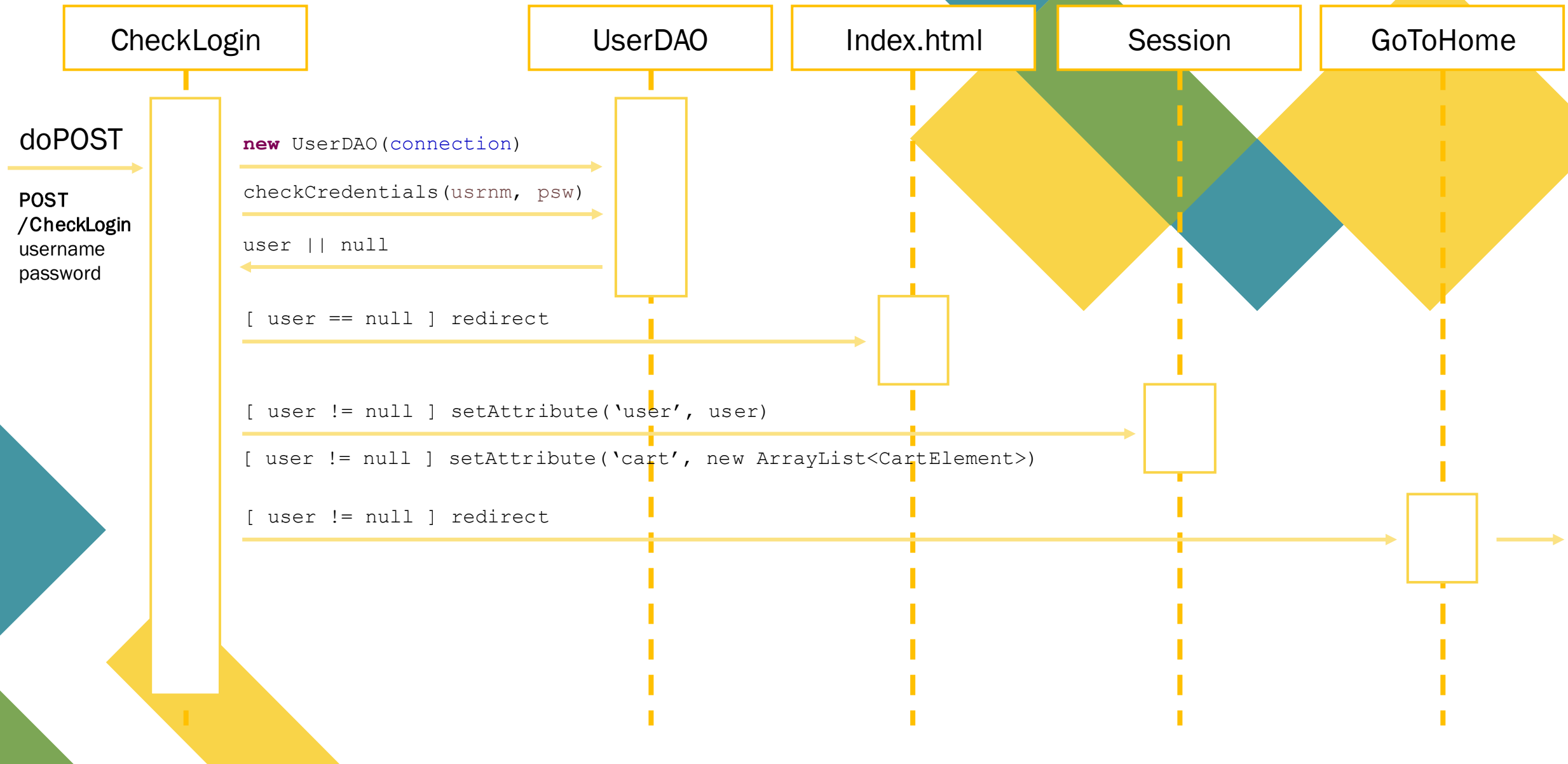
- *Model objects* (**Beans**)
  - User
  - Product
  - Supplier
  - PriceRange
  - Advertisement
  - CartElement
  - Order
- *Views* (**Templates**)
  - Menu.html
  - Home.html
  - Results.html
  - Cart.html
  - Orders.html
- *Data Access Objcts* (**DAO**)
  - UserDAO
    - `checkCredentials`
  - ProductDAO
    - `getProductDetails`
    - `getAdvDetails`
    - `getLastVisualizedByUser`
    - `getSearchedProducts`
    - `checkAvailability`
    - `buyProduct`
  - SupplierDAO
    - `getSupplierDetails`
    - `getAdvDetails`
    - `getAdvsSuppliersByProduct`
  - OrderDAO
    - `createOrder`
    - `getOrdersByUser`
- *Controllers* (**Servlets**)
  - Check Login
  - GoToHome
  - GoToResults
  - GoToCart
  - GoToOrders
  - AddToCart
  - RemoveFromCart
  - CreateOrder
  - Logout
- *Filters*
  - Checker
- *Utils*
  - ConnectionHandler

Sequence Diagrams

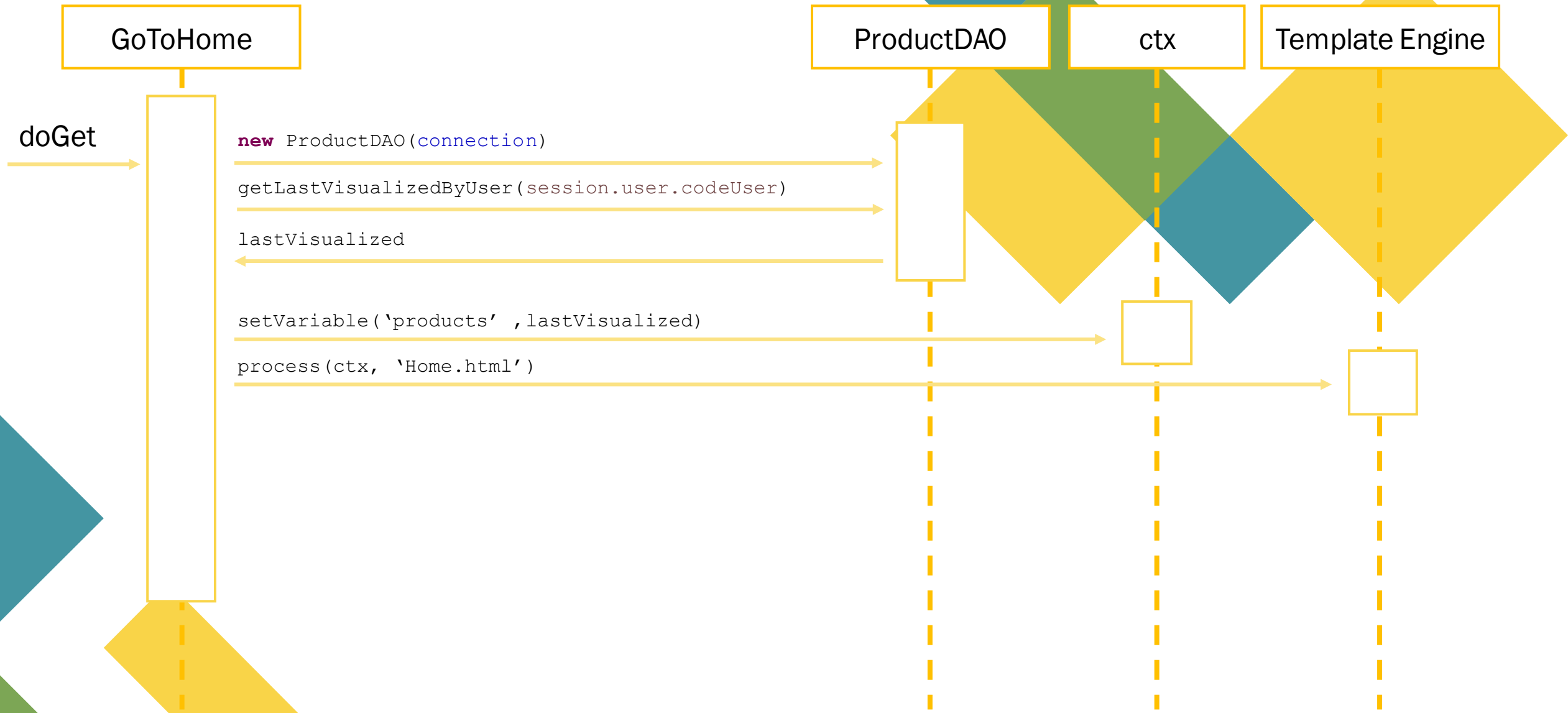
Prima di procedere con la presentazione dei sequence diagrams delle servlet adibite alla gestione della pagina, introduco la sezione con qualche appunto sulla documentazione:

- I box rossi all'interno dei sequence diagram definiscono dei blocchi condizionali. Quelli azzurri dei loop.
- Il sequence diagrams del filtro Checker viene omesso in quanto invariato rispetto quanto osservato a lezione
- Le servlet **AddToCart** e **RemoveFromCart** non dialogano con il database ma si limitano ad aggiornare l'istanza del carrello conservata nella sessione e a reindirizzare alla servlet **GoToCart**, pertanto sono state omesse per brevità.

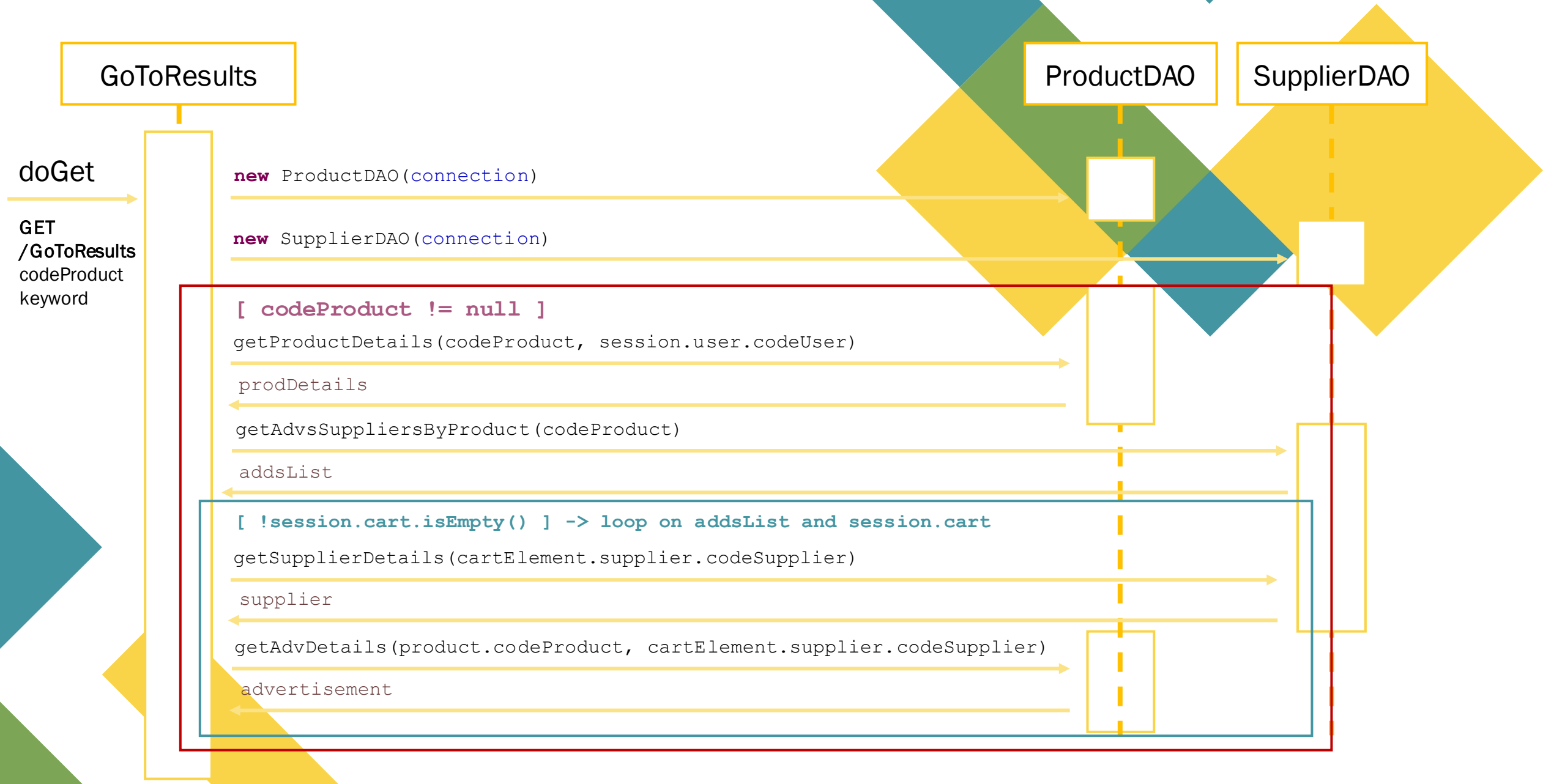
Event: Login

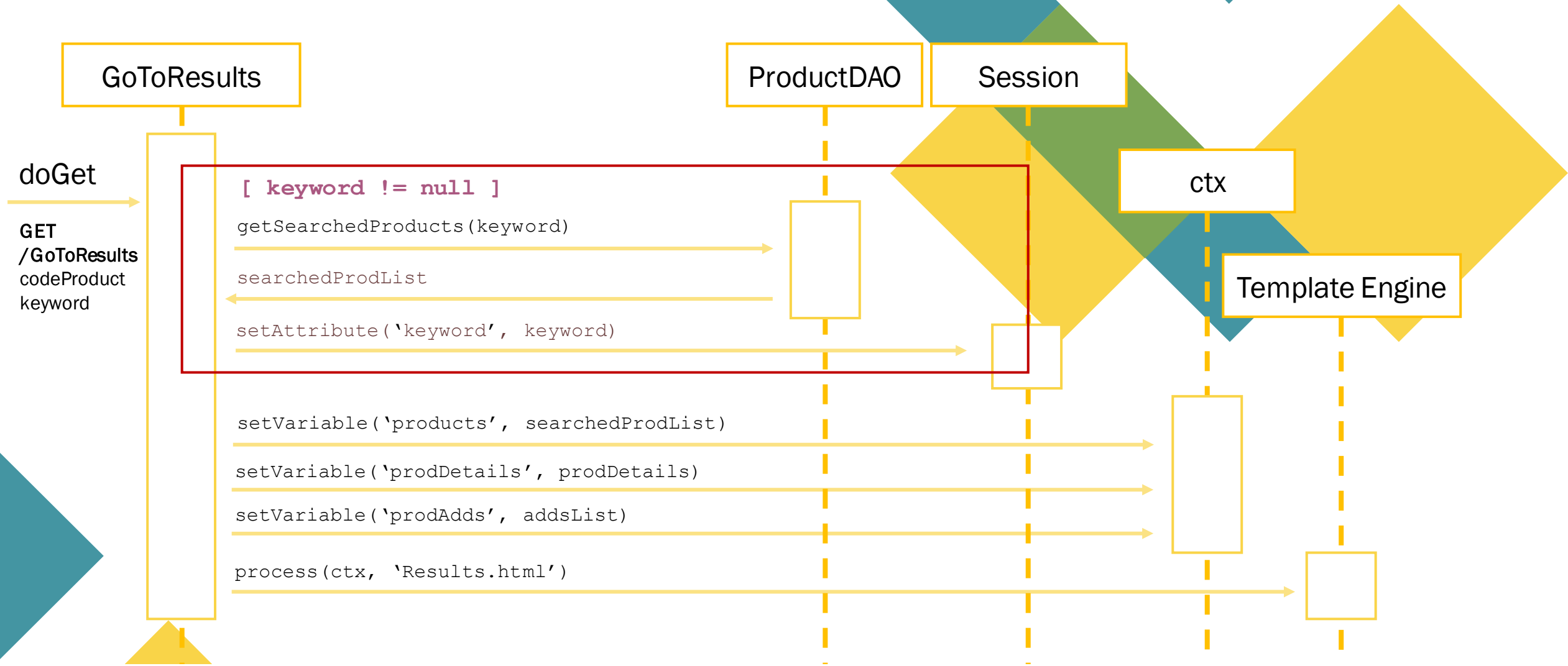

CheckLogin
UserDAO
Index.html
Session
GoToHome
doPOST
POST
/CheckLogin
username
password
new UserDAO(connection)
checkCredentials(usrnm, psw)
user || null
[user == null] redirect
[user != null] setAttribute('user', user)
[user != null] setAttribute('cart', new ArrayList<CartElement>)
[user != null] redirect

Event: Navigate to Home



Event: Navigate to Results

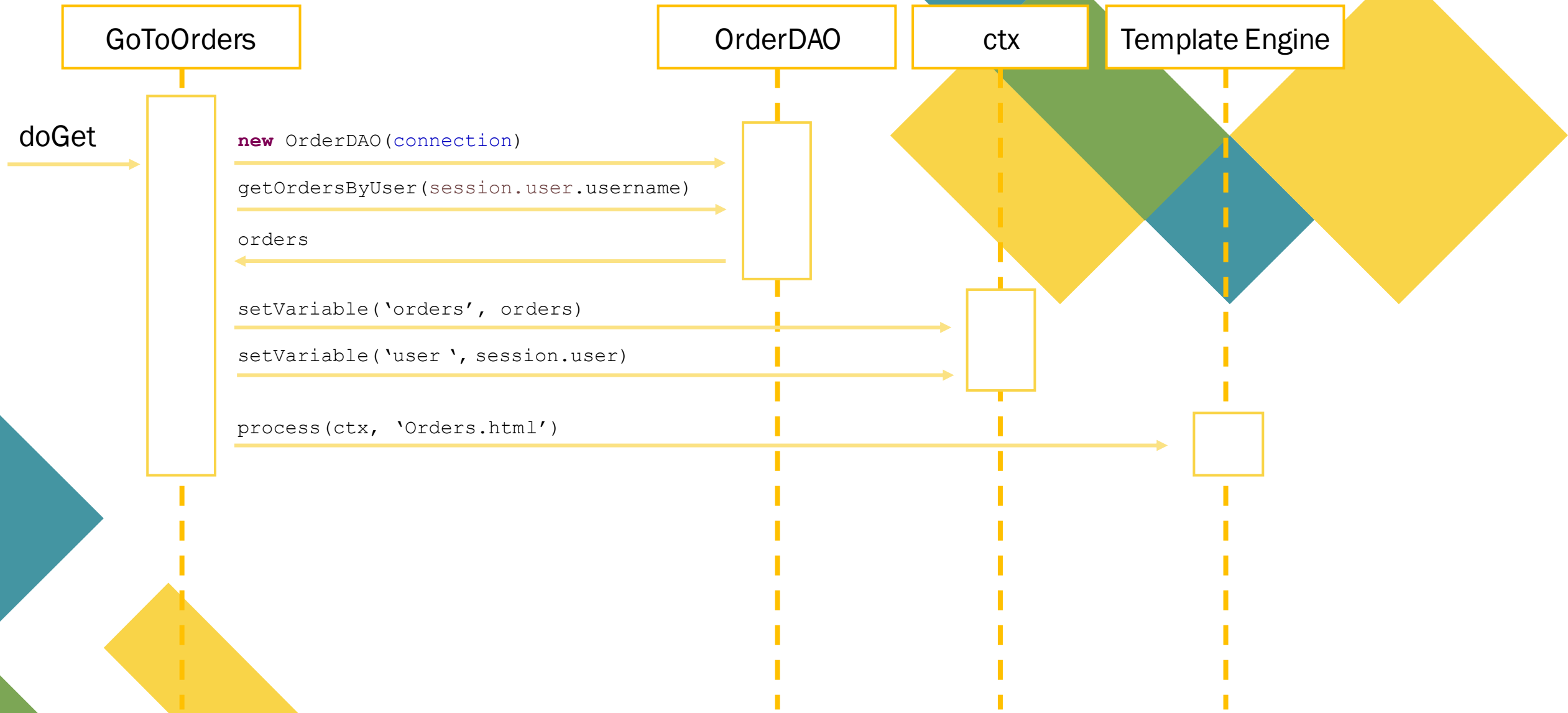




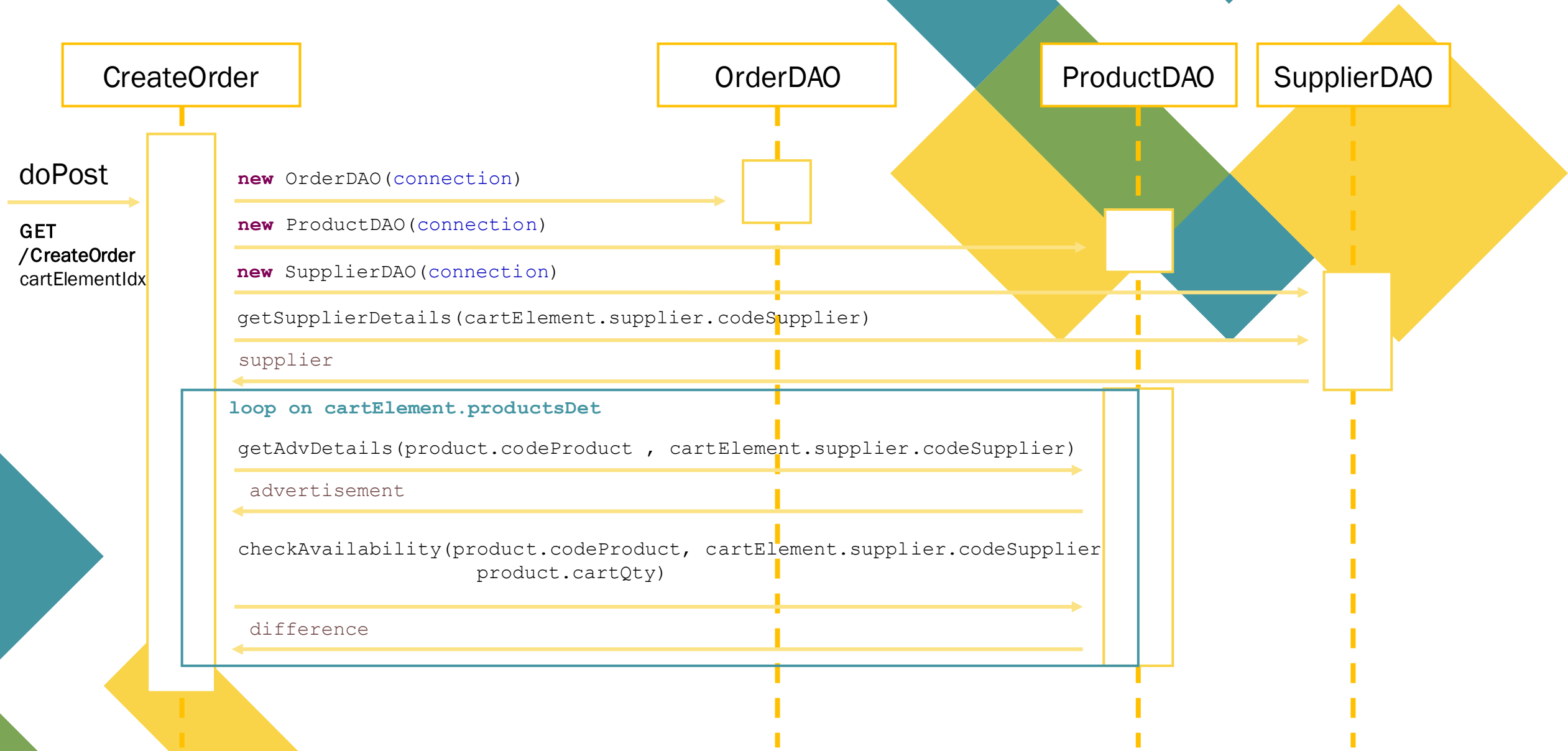
Keyword viene inserita all'interno della sessione al fine di evitare il ricaricamento della lista dei risultati nel caso in cui l'interazione sia provenuta dalla barra di ricerca.

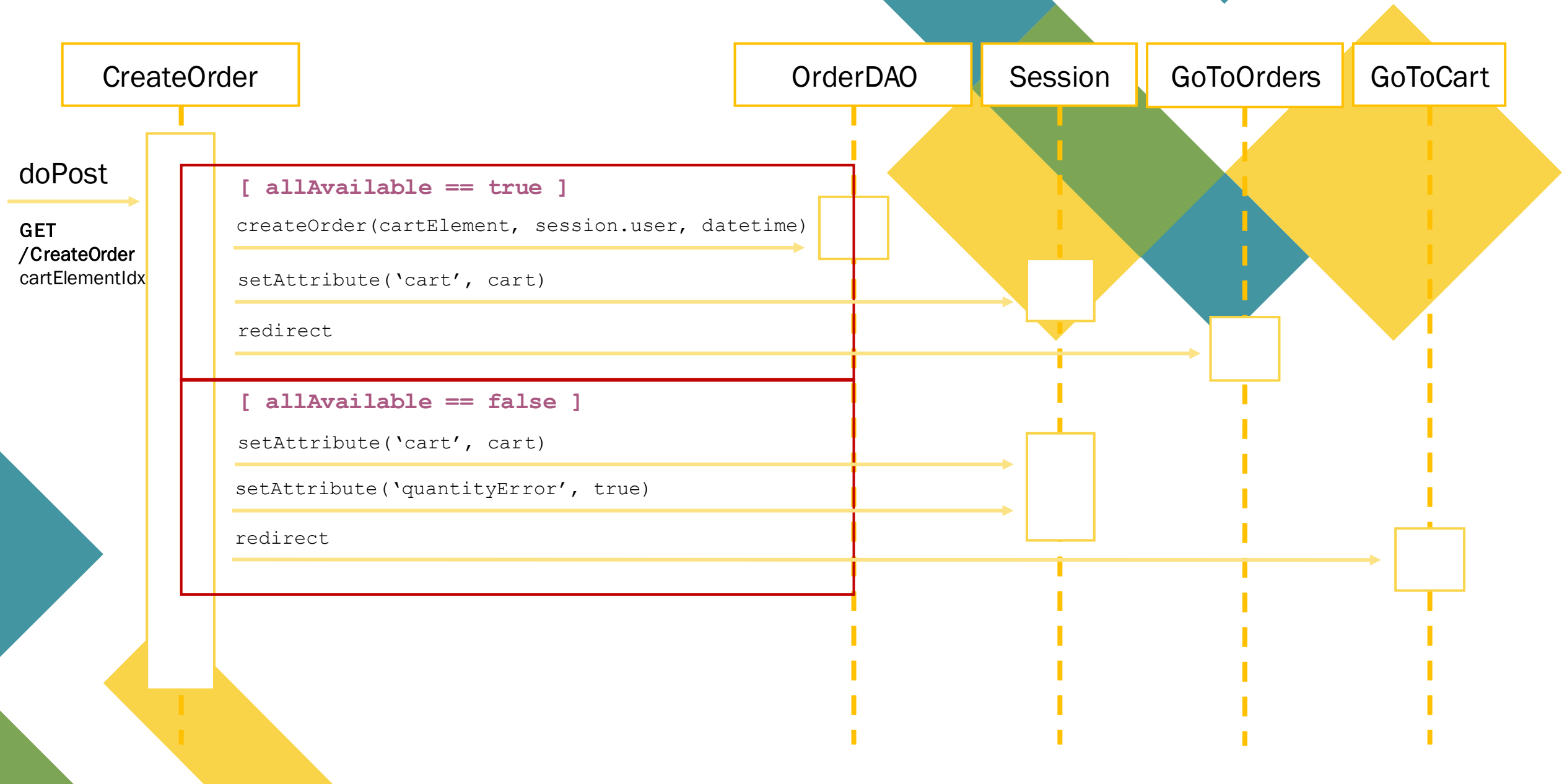
Il linking tra il carrello e la lista di annunci viene eseguito recuperando, per ogni elemento del carrello, i dati del fornitore e della sua politica di spedizione al fine di ricalcolare correttamente i prezzi di vendita e le quantità di elementi presenti nel carrello stesso.

Event: Navigate to Orders

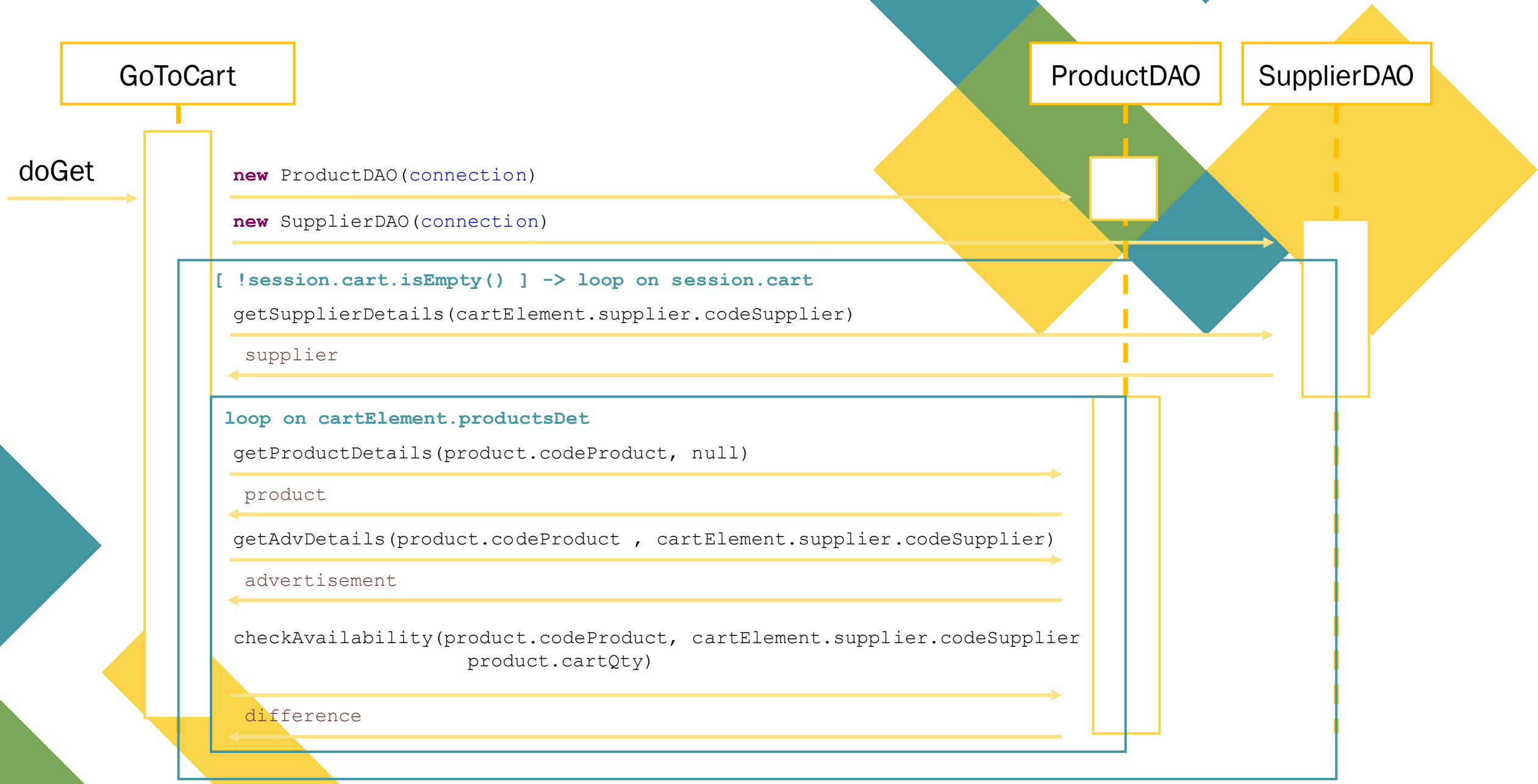


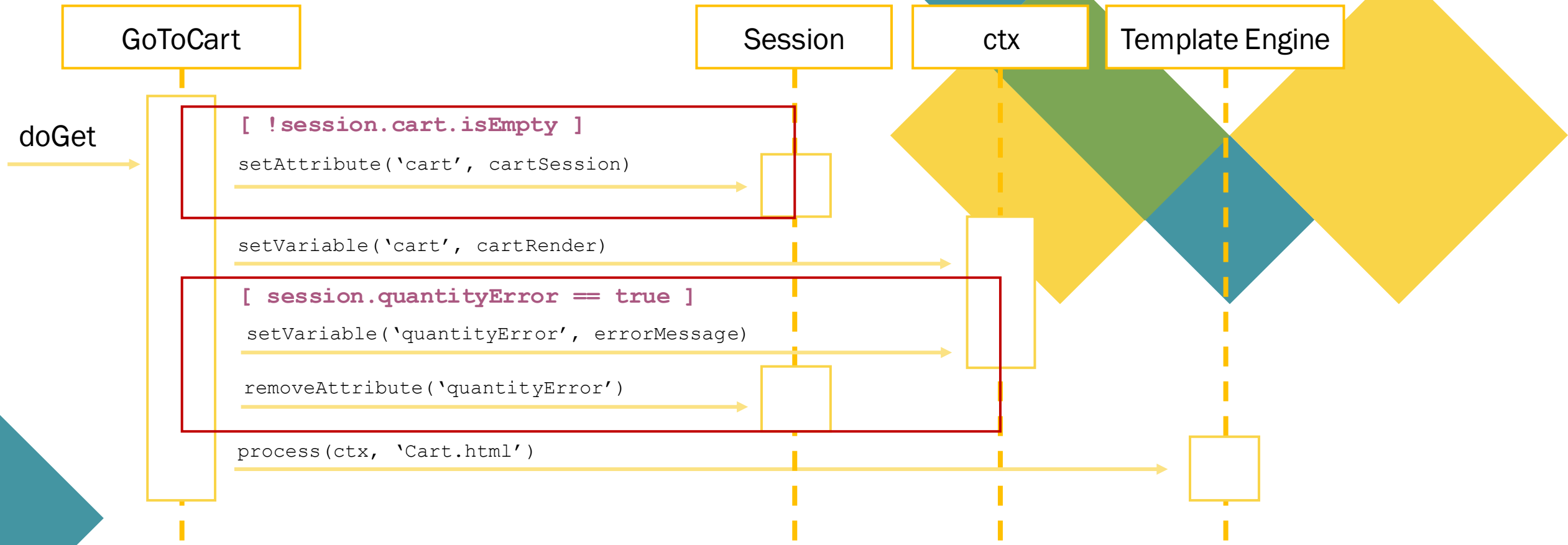
Event: Create Order




CreateOrder
OrderDAO
Session
GoToOrders
GoToCart
doPost
GET
/CreateOrder
cartElementIdx
[allAvailable == true]
createOrder(cartElement, session.user, datetime)
setAttribute('cart', cart)
redirect
[allAvailable == false]
setAttribute('cart', cart)
setAttribute('quantityError', true)
redirect

Event: Navigate to Cart

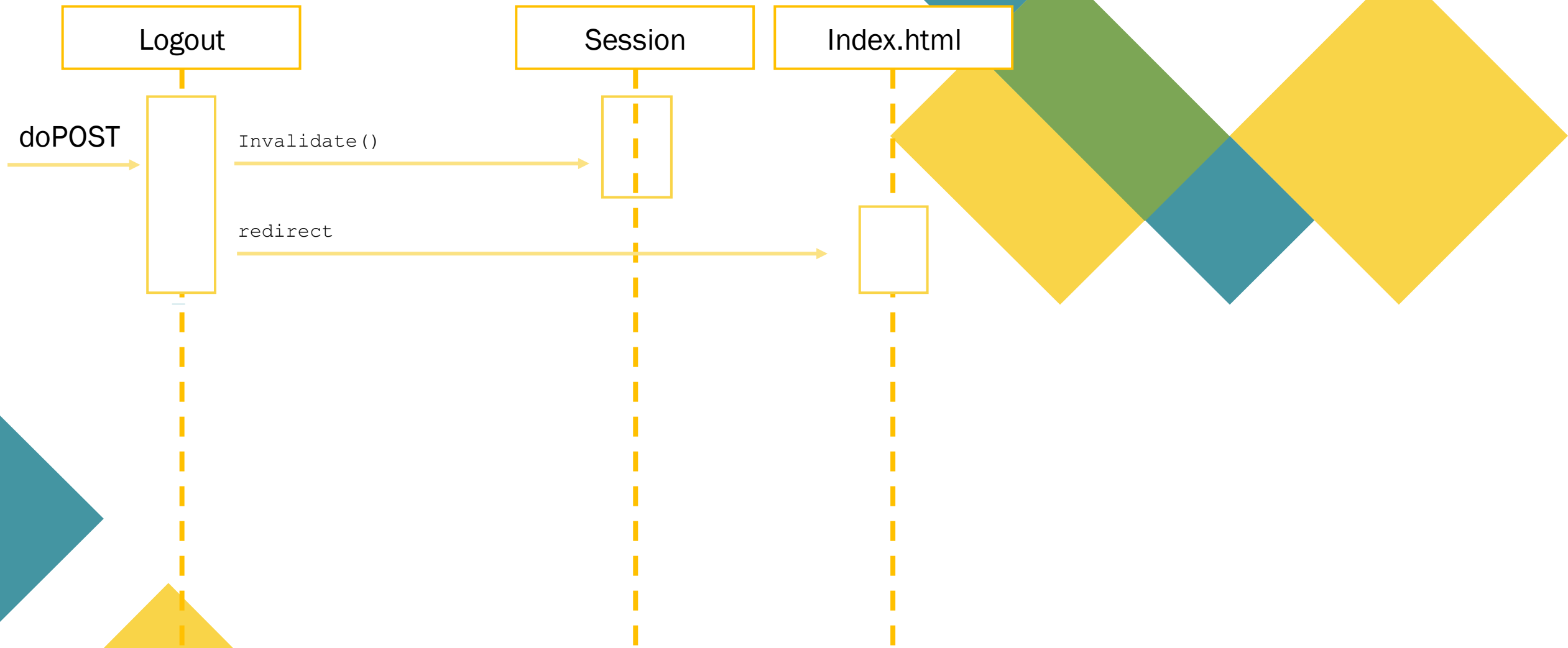




Il check sulla quantità presente nel carrello si basa sulla differenza tra la quantità attuale e la quantità presente nel database. Se tale differenza risulta maggiore di 0, la quantità nel carrello è maggiore di quella nel database, pertanto si procede a sanificare il valore sottraendogli la differenza stessa al fine di pareggiare il risultato con «al più» il valore nel database.
In caso la quantità del prodotto venga portata a 0, un flag segnala il prodotto come «out of stock» La rimozione di tale prodotto dal carrello avviene in un secondo momento.

Il messaggio di errore renderizzato in caso almeno un prodotto non sia più disponibile recita:
«Some quantities inside your cart have been changed due to limited supply in stock»

Event: Logout



Versione RIA

Differenze e similitudini...

Le caratteristiche che differenziano, a livello di logica implementativa, la versione pureHTML dalla versione RIA riguardano principalmente la gestione del carrello.
Quest'ultimo, nella versione RIA, non è più memorizzato nella sessione a lato server, ma viene mantenuto all'interno del sessionStorage a lato client, come richiesto nella specifica.

Onde evitare anche in questa versione asincronie con il database, qualsiasi azione che vede necessario l'aggiornamento del cart-client viene svolta a lato server e prevede il recupero di tale oggetto dal body della request interessata per poi restituire l'oggetto aggiornato sotto forma di json.

La funzione ***makeCall*** è stata, per necessità, rivisitata ma non sostituita da una funzione ***makeJsonCall*** che prevede la possibilità di inviare un oggetto javascript all'interno della request. Tale oggetto viene poi estratto nella servlet interessata servendosi della libreria Gson e gestito congruentemente al suo scopo.

La scelta di gestire l'aggiunta e la rimozione di elementi dal carrello a lato server è stata maturata per ridurre ulteriormente la possibilità che un utente malevolo possa mettere mano allo script e inserire all'interno del carrello elementi problematici.

View Components

- *HomeCS.html*
  - Menù
  - Sezione Home
    - Lista prodotti visualizzati di recente
  - Sezione Risultati
    - Lista prodotti ricercati
    - Dettagli prodotto
    - Lista Annunci prodotto
  - Sezione Ordini
    - Lista ordini
  - Sezione Carrello
    - Lista elementi nel carrello

- *HoverCS.html*
  - Lista prodotti nel carrello dallo stesso fornitore

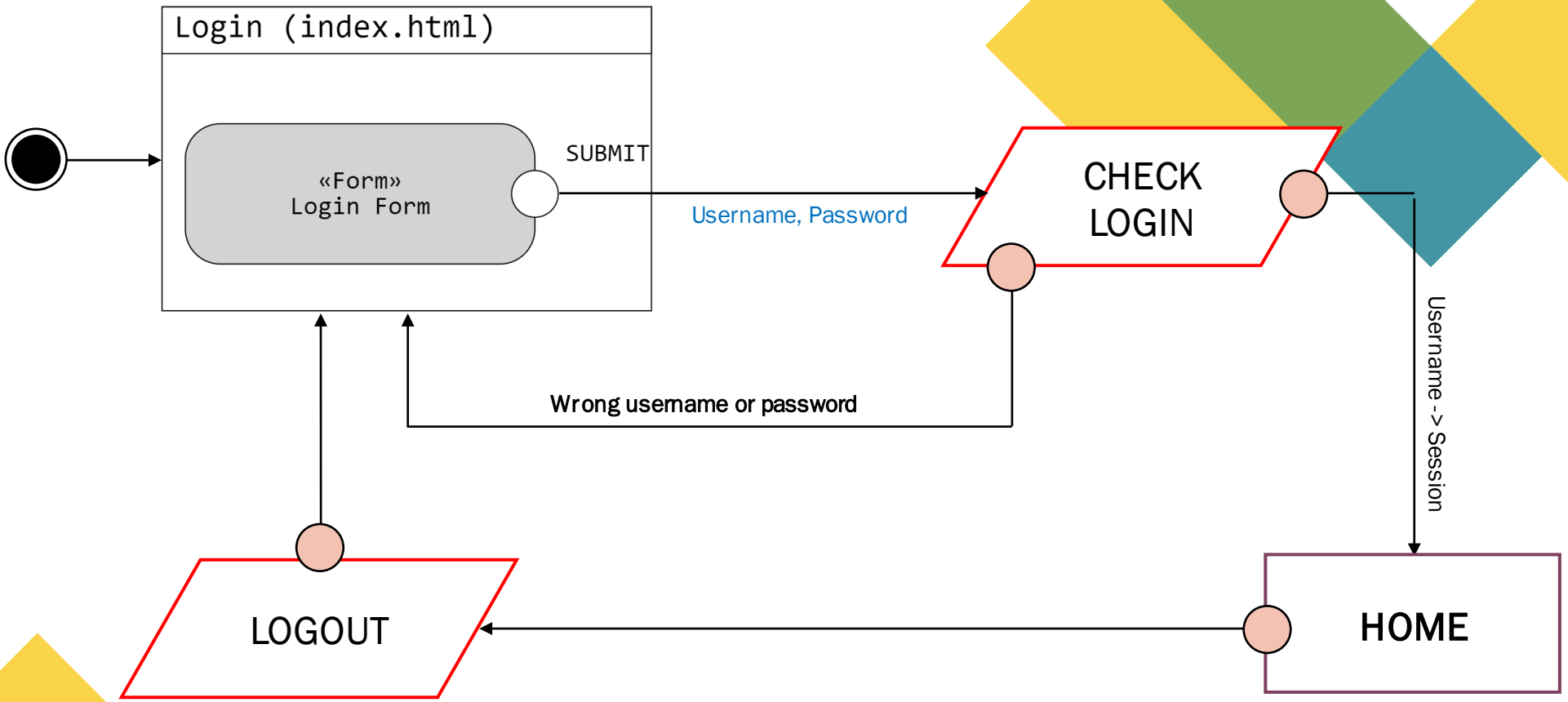
Tutti i componenti nella view sono inizializzati dal PageManager durante l'evento «***load***».

Una funzione «***show***» determina dinamicamente quali sezioni sono o meno renderizzate all'interno del DOM in base alle scelte dell'utente.

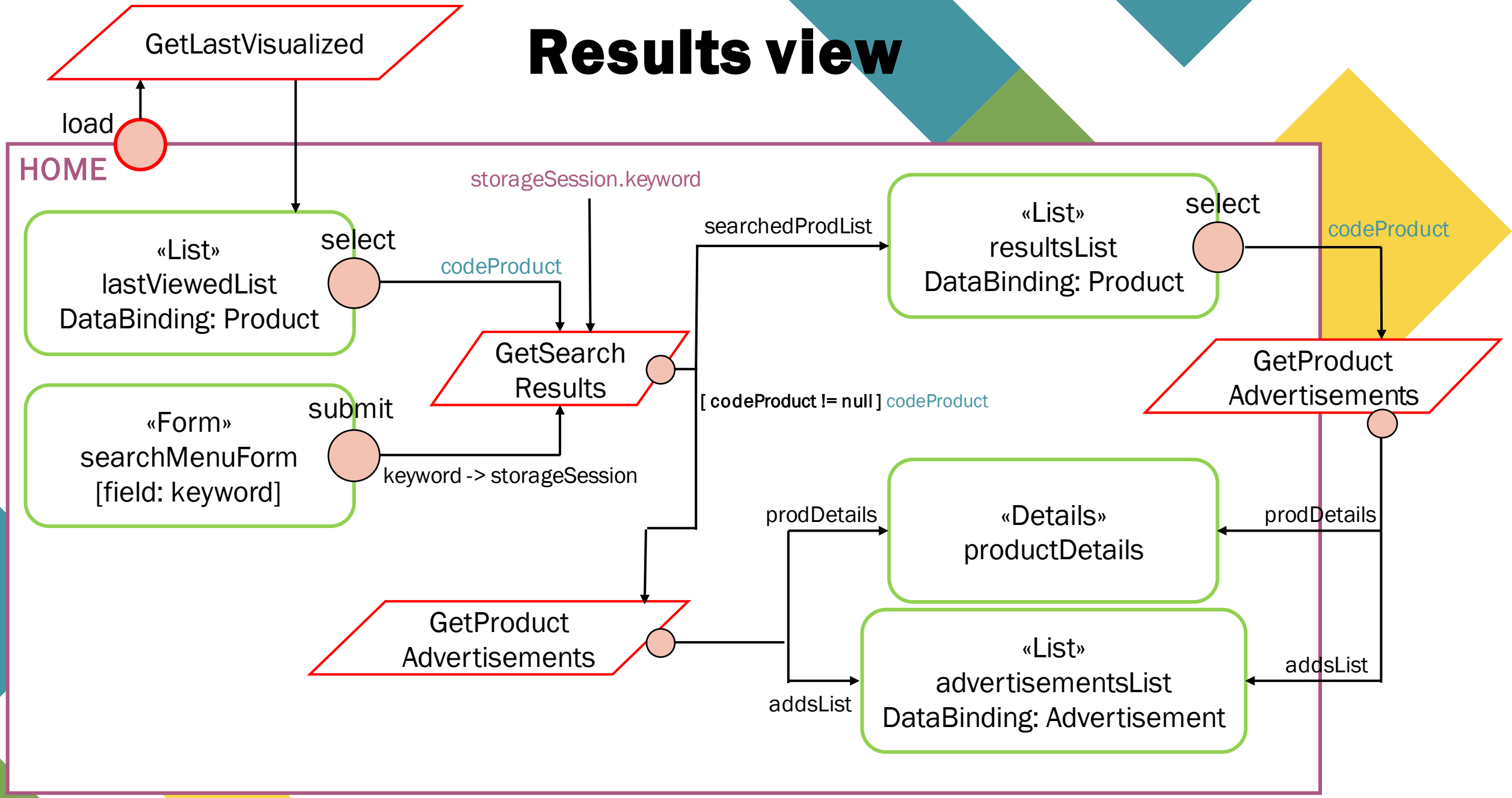
Funzioni interne ai componenti determinano quali elementi di tali sezioni sono renderizzati nel DOM.

Onde evitare che l'utente possa svolgere qualche azione impropria tramite l'inspector del browser, le sezioni «inattive» vengono prontamente svuotate del loro contenuto ad ogni reset della pagina.

Application Design (IFML)

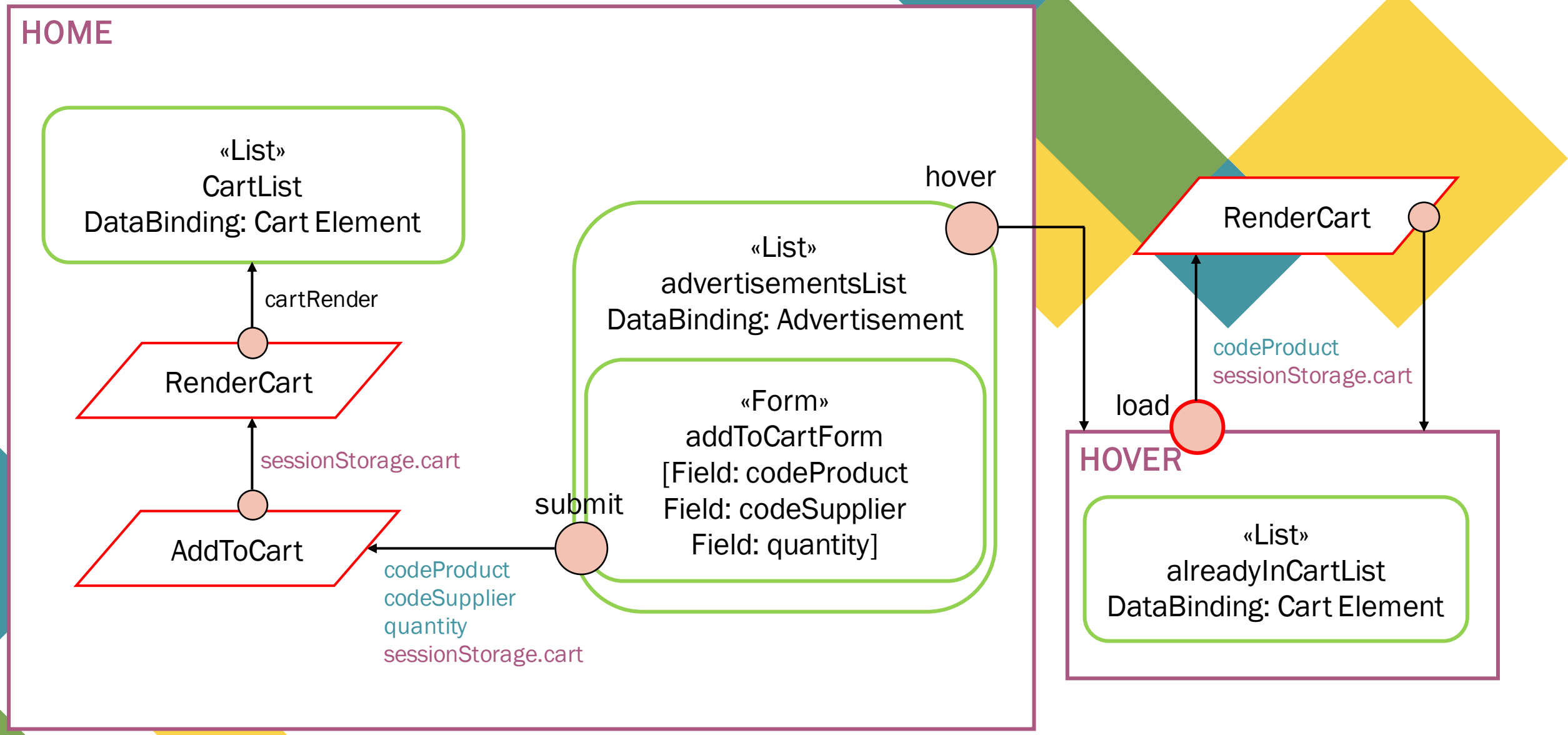


Results view



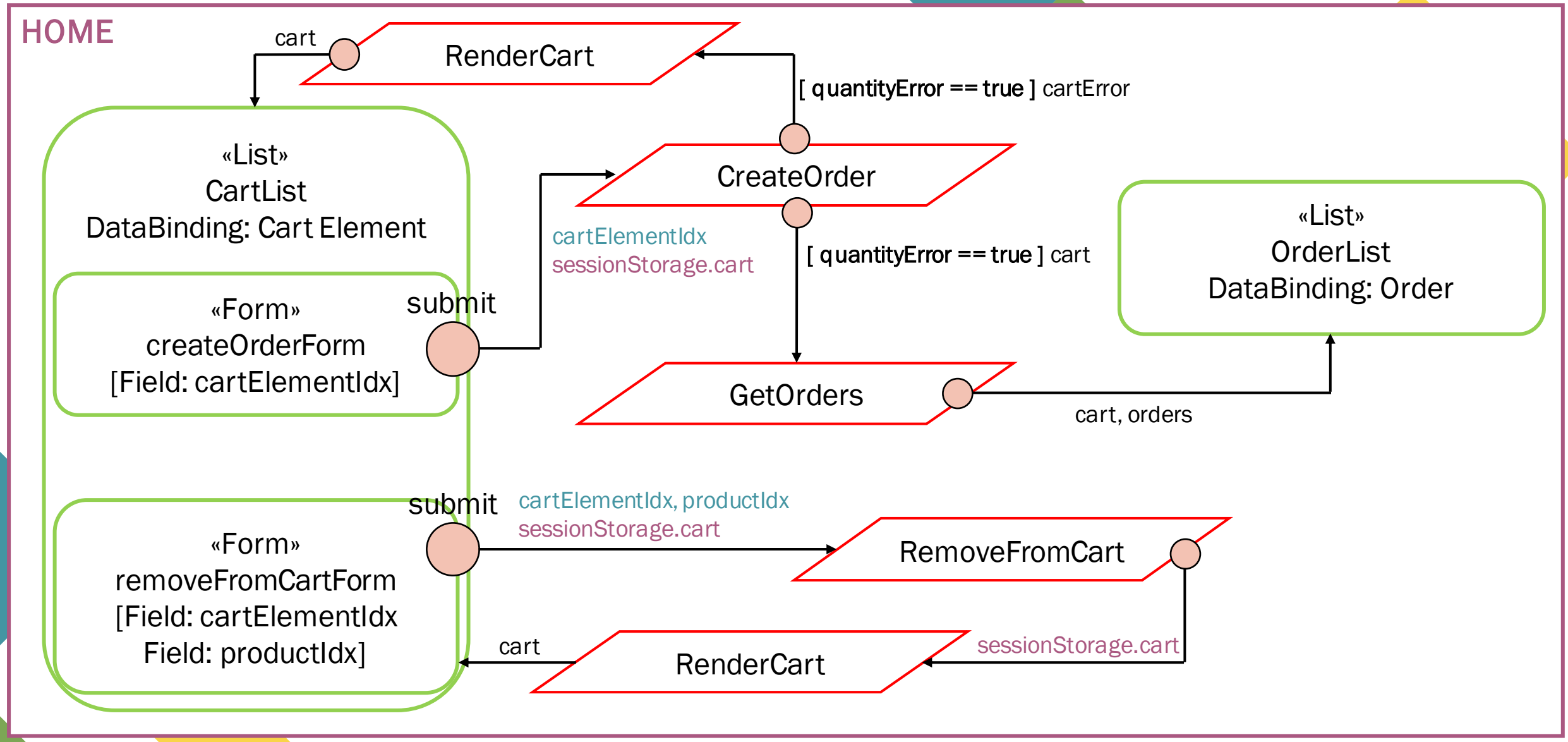
Gli eventi relativi alle ancore nel menù sono stati omessi per brevità

Cart view



Gli eventi relativi alle ancore nel menù sono stati omessi per brevità

Order view



Gli eventi relativi alle ancore nel menù sono stati omessi per brevità

Eventi e Azioni

Gli eventi relativi alle ancore nel menù sono stati omessi per brevità

Client Side		Server Side	
Evento	Azione	Evento	Azione
Index.html -> login Form -> Submit	Controllo validità form	POST [username, password]	Controllo credenziali
HomeCS.html -> Load	Aggiornamento view con lista prodotti visualizzati di recente	GET []	Estrazione ultimi 5 visualizzati dall'utente
HomeCS.html -> search Form -> Submit	Aggiornamento view con lista prodotti ricercati	GET [keyword]	Estrazione prodotti con parola chiave nel nome o nella descrizione
HomeCS.html -> last Visualized List -> Select	Aggiornamento view con lista prodotti ricercati, dettagli e annunci del prodotto selezionato + link con carrello	GET [codeProduct] ↓ POST [{storageSession.cart, codeProduct}]	Aggiunta prodotto selezionato alla lista risultati Estrazione dettagli e annunci del prodotto selezionato
HomeCS.html -> result List -> Select	Aggiornamento view con dettagli e annunci del prodotto selezionato + link con carrello	POST [{storageSession.cart, codeProduct}]	Estrazione dettagli e annunci del prodotto selezionato
HomeCS.html -> advertisements List -> add To Cart Form -> Submit	Aggiornamento view con lista prodotti nel carrello	POST [{storageSession.cart. codeProduct, codeSupplier, quantity}] ↓ POST [storageSession.cart]	Aggiunta al carrello del prodotto nella quantità scelta. Controllo di congruenza degli elementi del carrello con le quantità in stock.

Client Side		Server Side	
Evento	Azione	Evento	Azione
HomeCS.html -> advertisements List -> Hover	Apertura di una nuova finestra nel browser -> /HoverCS.html	-	-
HoverCS.html -> Load	Aggiornamento view con lista prodotti già nel carrello dallo stesso fornitore	POST [{storageSession.cart, codeSupplier}]	Estrazione dal carrello della lista di prodotti del fornitore Controllo di congruenza degli elementi del carrello con le quantità in stock.
HomeCS.html -> cart List -> remove From Cart Button -> Click	Aggiornamento view con lista prodotti nel carrello	POST [{storageSession.cart. cartElementIdx, productIdx}] ↓ POST [storageSession.cart]	Rimozione dal carrello del prodotto. Controllo di congruenza degli elementi del carrello con le quantità in stock
HomeCS.html -> cart List -> create Order Button -> Click	Aggiornamento view con lista ordini OR Aggiornamento view con carrello aggiornato	POST [{storageSession.cart. cartElementIdx}] ↓ GET [] OR POST [storageSession.cart]	Controllo di congruenza degli elementi del carrello con le quantità in stock Rimozione dal carrello del prodotto e caricamento ordini. OR Aggiornamento carrello caricamento nuovo carrello

Controller / Event Handler

Gli eventi relativi alle ancore nel menù sono stati omessi per brevità

Client Side		Server Side	
Evento	Azione	Evento	Azione
Index.html -> login Form -> Submit	Function makeCall	POST [username, password]	CheckLogin (Servlet)
HomeCS.html -> Load	Function PageManager.start()	GET []	/GetLastVisualized (Servlet)
HomeCS.html -> search Form -> Submit	Function makeCall	GET [keyword]	/GetSearchResults (Servlet)
HomeCS.html -> last Visualized List -> Select	Function makeCall ↓ Function makeJsonCall	GET [codeProduct] ↓ POST [{storageSession.cart, codeProduct}]	/GetSearchResults (Servlet) ↓ /GetProductAdvertisements (Servlet)
HomeCS.html -> result List -> Select	Function makeJsonCall	POST [{storageSession.cart, codeProduct}]	/GetProductAdvertisements (Servlet)
HomeCS.html -> advertisements List -> add To Cart Form -> Submit	Function makeJsonCall ↓ Function makeJsonCall	POST [{storageSession.cart. codeProduct, codeSupplier, quantity}] ↓ POST [storageSession.cart]	/AddToCart (Servlet) ↓ /RenderCart (Servlet)

Client Side		Server Side	
Evento	Azione	Evento	Azione
HomeCS.html -> advertisements List -> Hover	Window.open(...)	-	-
HoverCS.html -> Load	Function makeJsonCall	POST [{storageSession.cart, codeSupplier}]	/RenderCart (Servlet)
HomeCS.html -> cart List -> remove From Cart Button -> Click	makeJsonCall ↓ makeJsonCall	POST [{storageSession.cart. cartElementIdx, productIdx}] ↓ POST [storageSession.cart]	/RemoveFromCart (Servlet) ↓ /RenderCart (Servlet)
HomeCS.html -> cart List -> create Order Button -> Click	makeJsonCall ↓ makeCall OR makeJsonCall	POST [{storageSession.cart. cartElementIdx}] ↓ GET [] OR POST [storageSession.cart]	/createOrder (Servlet) ↓ /getOrders (Servlet) OR /RenderCart (Servlet)

Server side components

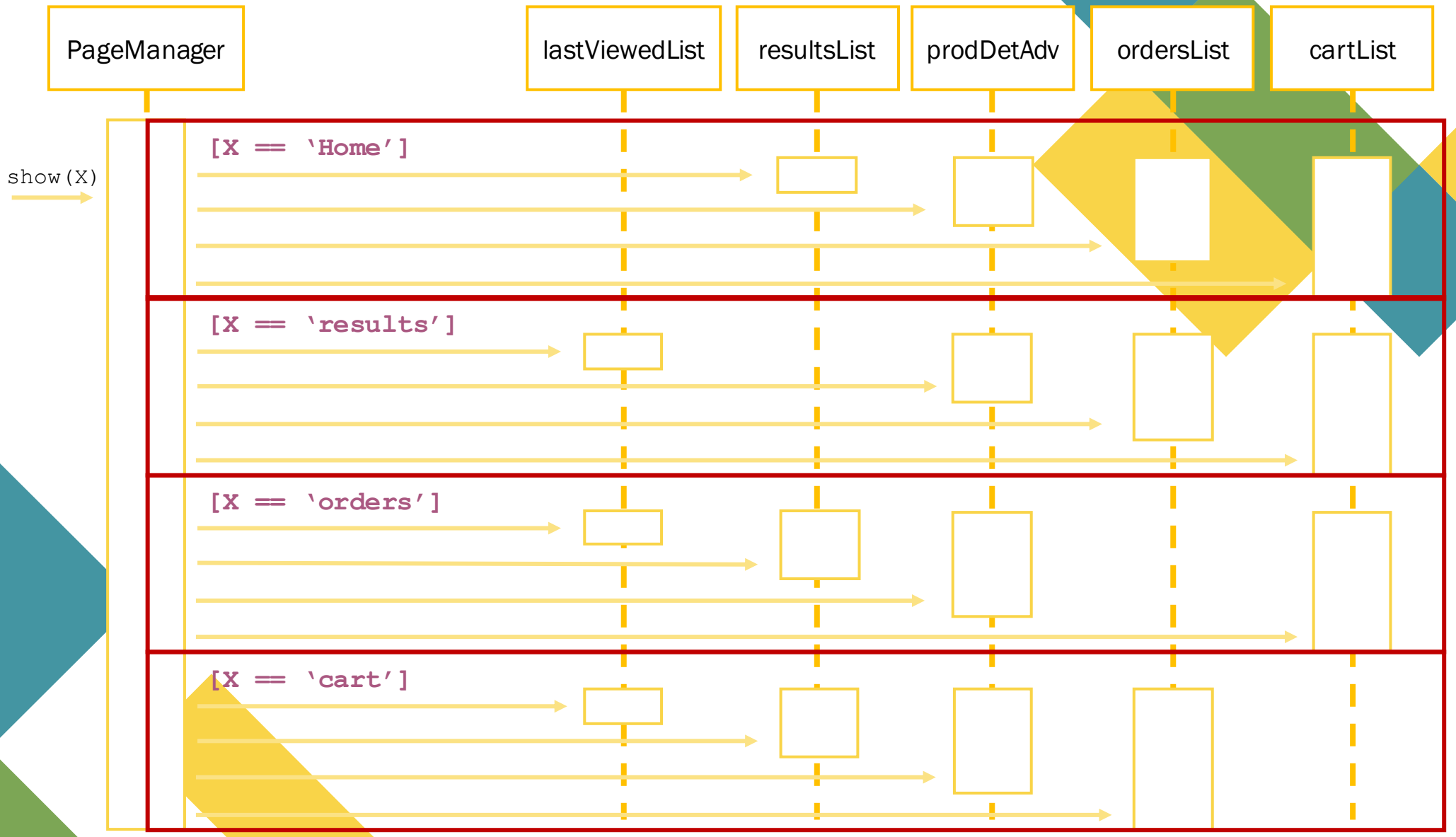
- *Model objects* (**Beans**)
  - User
  - Product
  - Supplier
  - PriceRange
  - Advertisement
  - CartElement
  - Order
- *Views* (**Templates**)
  - HomeCS.html
  - HoverCS.html
- *Data Access Objcts* (**DAO**)
  - UserDAO
    - `checkCredentials`
  - ProductDAO
    - `getProductDetails`
    - `getAdvDetails`
    - `getLastVisualizedByUser`
    - `getSearchedProducts`
    - `checkAvailability`
    - `buyProduct`
  - SupplierDAO
    - `getSupplierDetails`
    - `getAdvDetails`
    - `getAdvsSuppliersByProduct`
  - OrderDAO
    - `createOrder`
    - `getOrdersByUser`
- *Controllers* (**Servlets**)
  - Check Login
  - GetLastVisualized
  - GetSearchResults
  - GetProductAdvertisements
  - RenderCart
  - AddToCart
  - RemoveFromCart
  - GetOrders
  - CreateOrder
  - Logout
- *Filters*
  - Checker
  - NoCacher
- *Utils*
  - ConnectionHandler

Sequence Diagrams

Prima di procedere con la presentazione dei sequence diagrams delle servlet adibite alla gestione della pagina, introduco la sezione con qualche appunto sulla documentazione:

- I box rossi all'interno dei sequence diagram definiscono dei blocchi condizionali. Quelli azzurri dei loop.
- I sequence diagrams dei filtri **Checker** e **NoCacher** vengono omessi in quanto invariati rispetto quanto osservato a lezione.
- I sequence diagrams delle servlet **CreateOrder** e **GetOrders** vengono omessi in quanto invariati rispetto alla versione pureHTML
- La funzione **show**() del PageManager viene analizzata a parte e non viene inclusa nei sequence diegram per chiarezza.
- Client Side Server Side

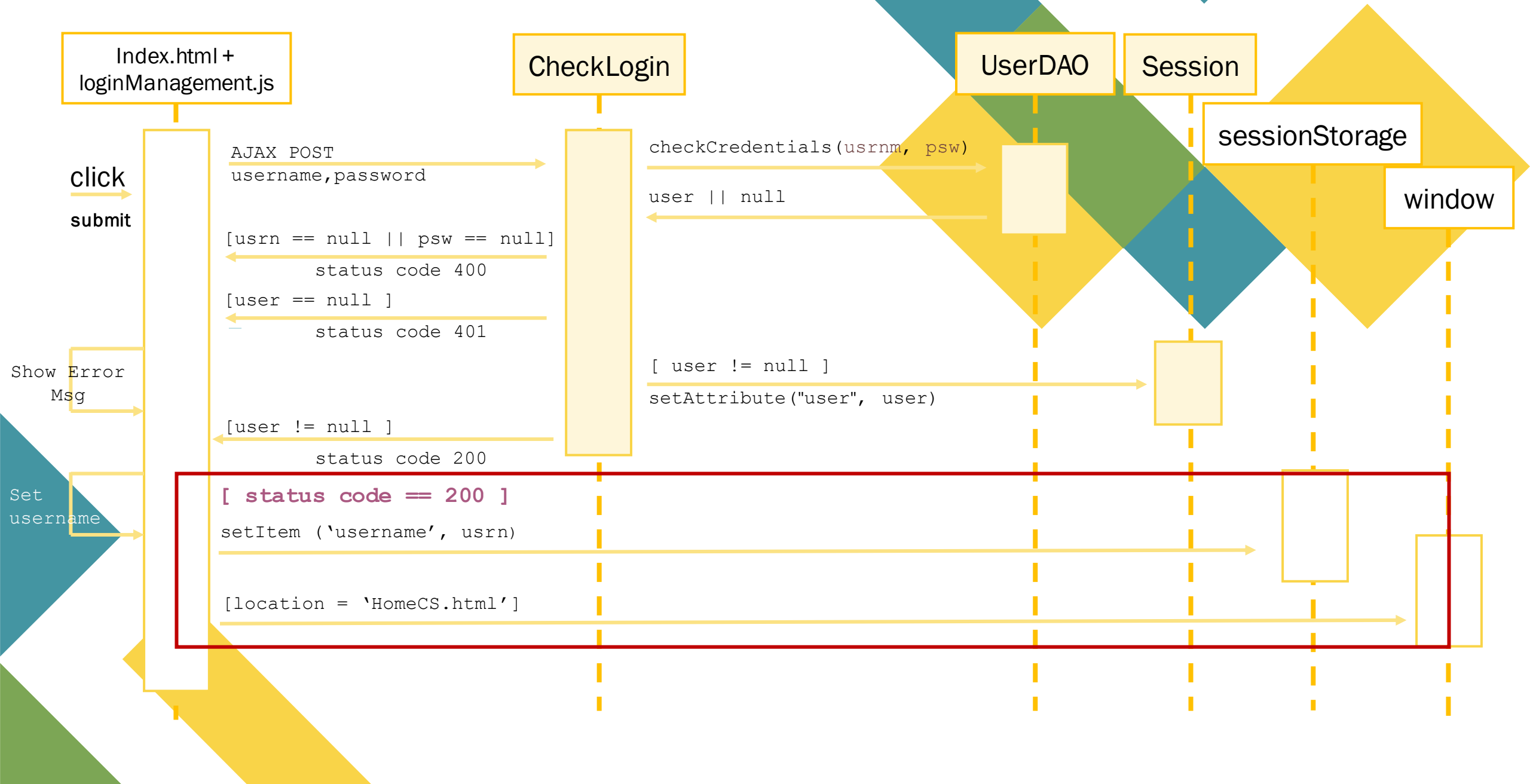
Function: pageManager.show(X)



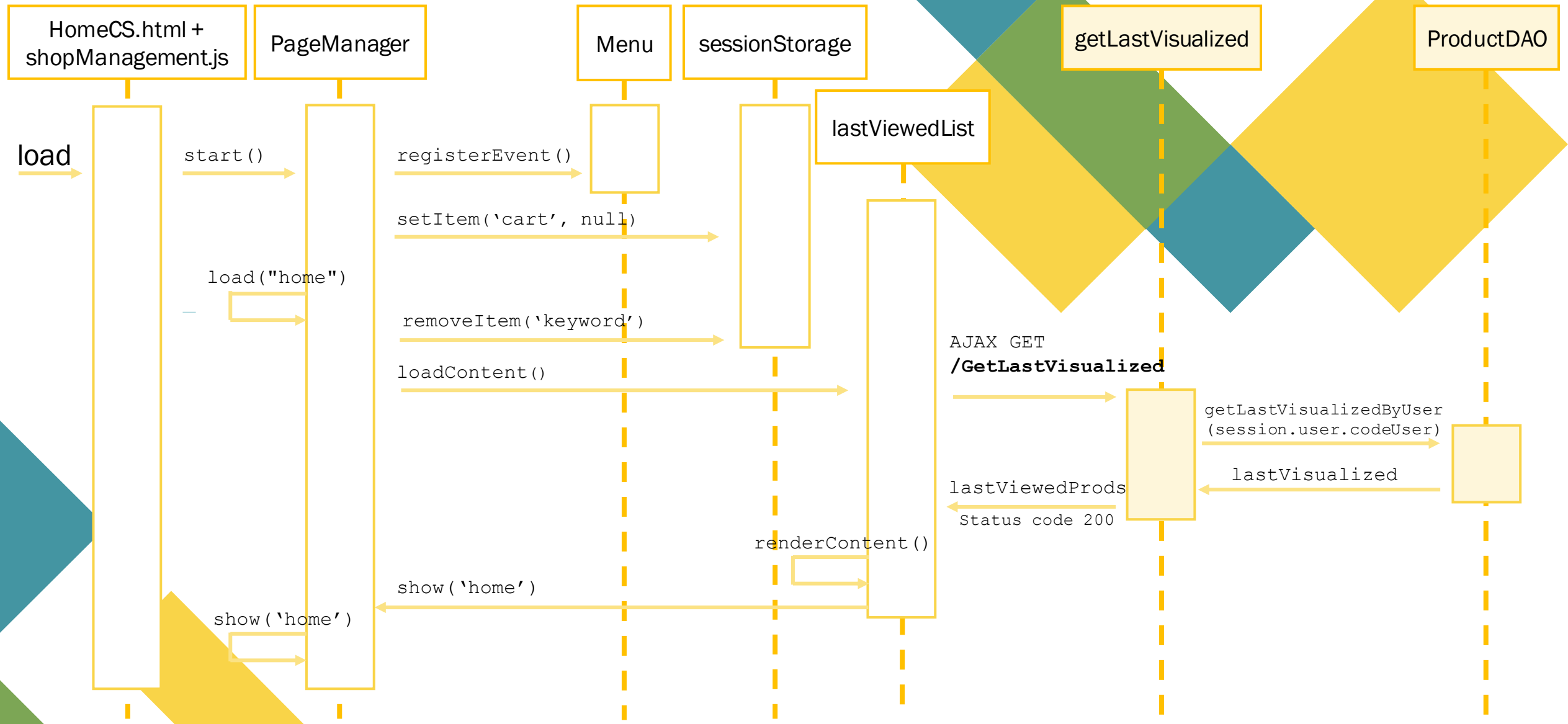
La funzione richiamata in ogni interazione è *component*.**resetContent**()

Il suo obiettivo è quello di resettare il contenuto del componente al fine di impedire manomissioni da parte dell'utente e l'accesso a funzionalità non disponibili nella view attuale.

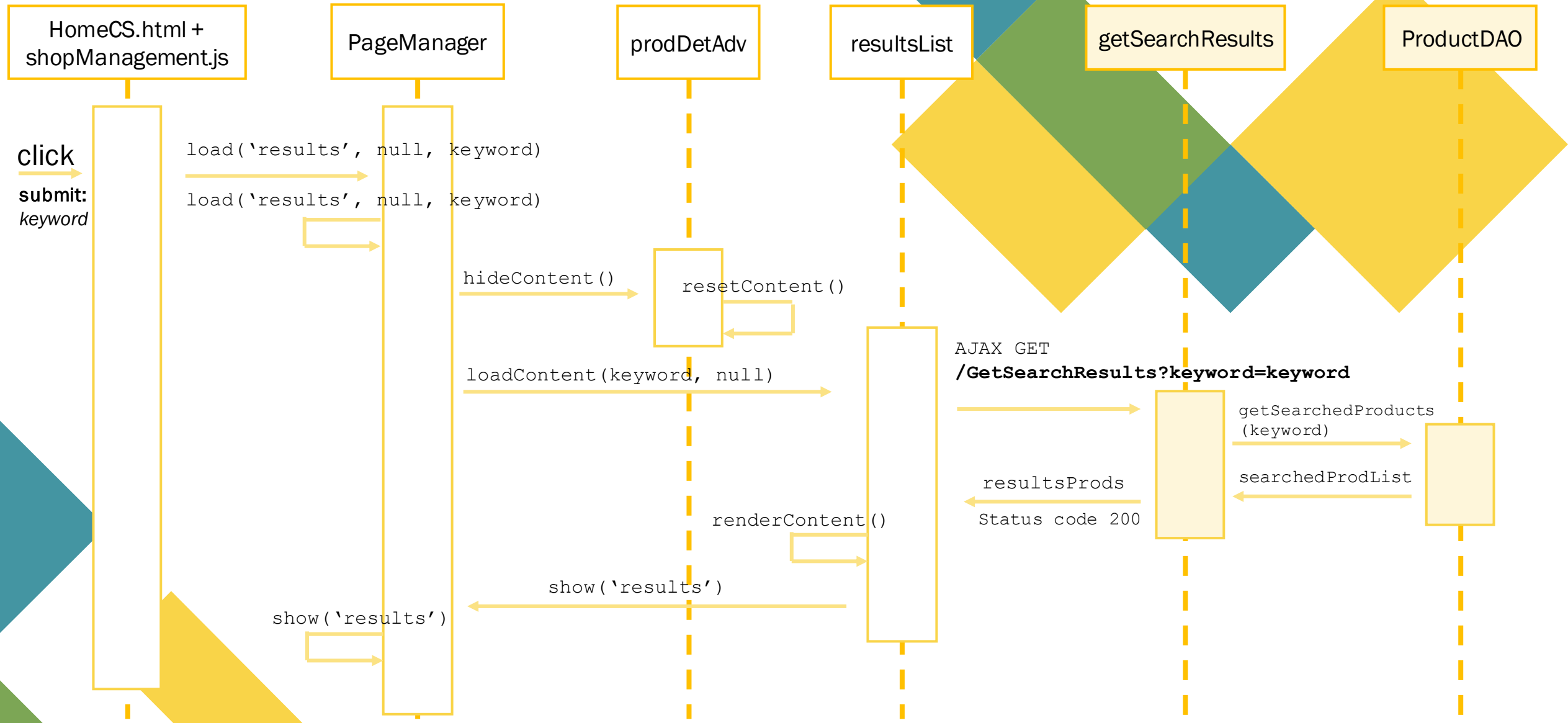
Event: Login


Index.html + loginManagement.js
CheckLogin
UserDAO
Session
sessionStorage
window
click
submit
AJAX POST
username,password
checkCredentials(usrnm, psw)
user || null
[usrn == null || psw == null]
status code 400
[user == null]
status code 401
Show Error Msg
[user != null]
setAttribute("user", user)
[user != null]
status code 200
Set username
[status code == 200]
setItem ('username', usrn)
[location = 'HomeCS.html']

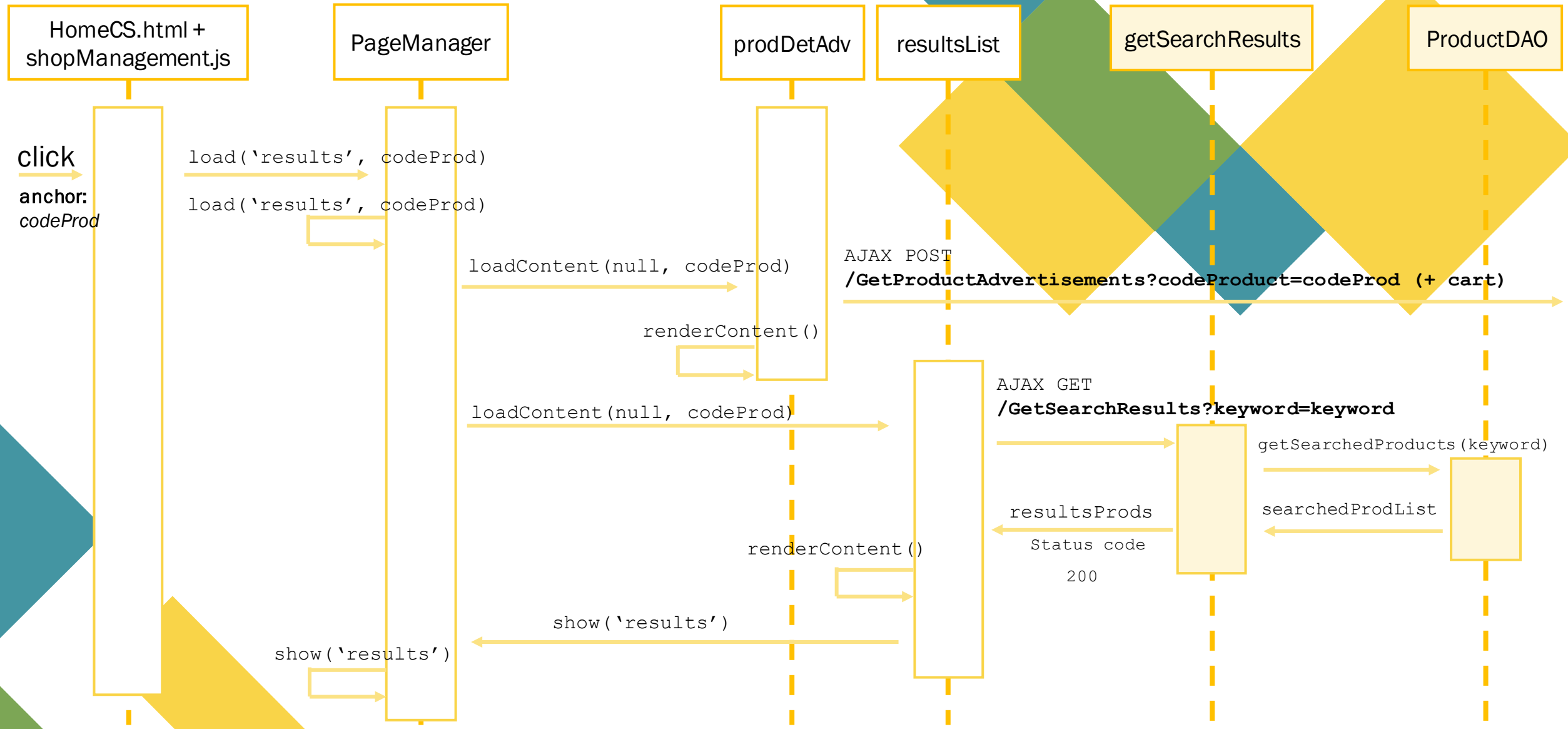
Event: Load HomeCS.html



Event: Load Results from searchForm

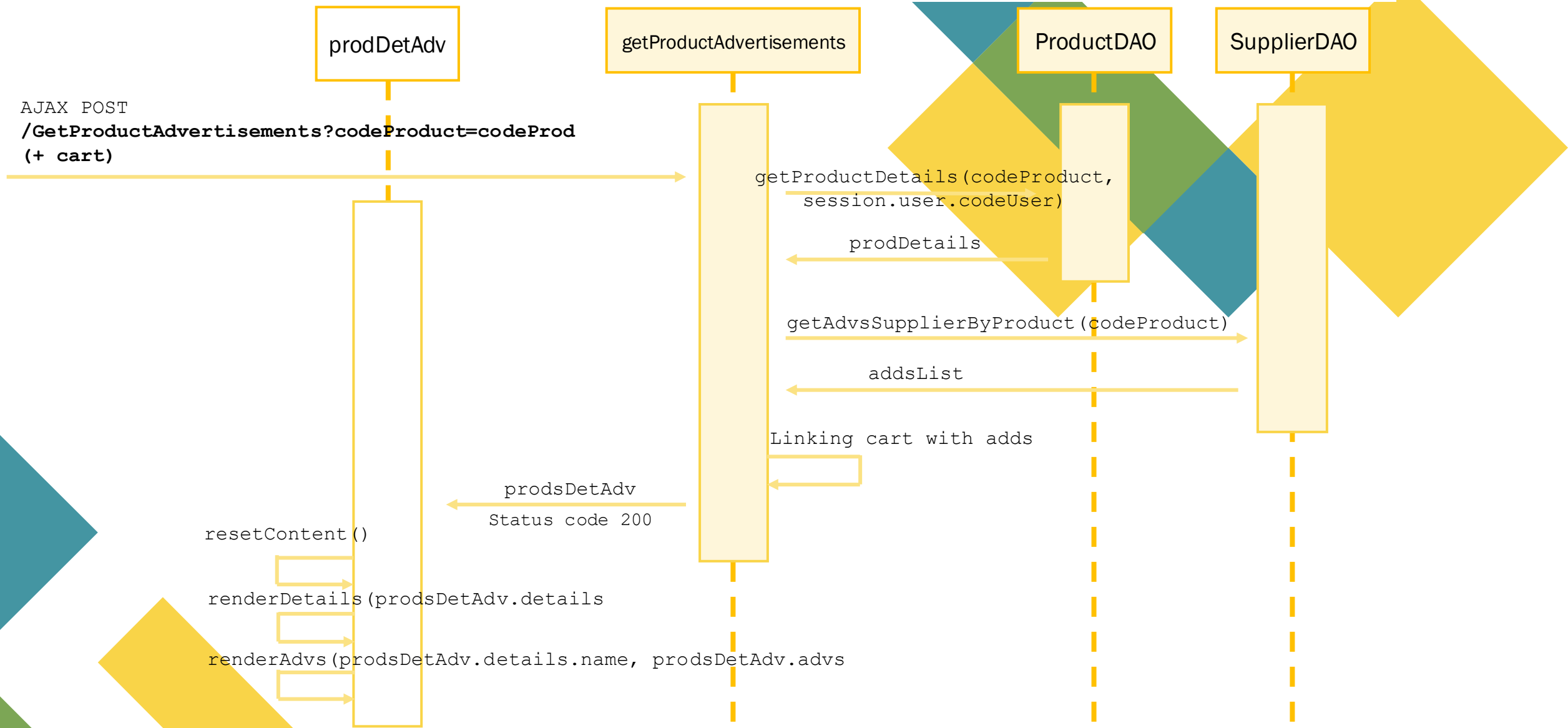


Event: Load Results from lastViewedList

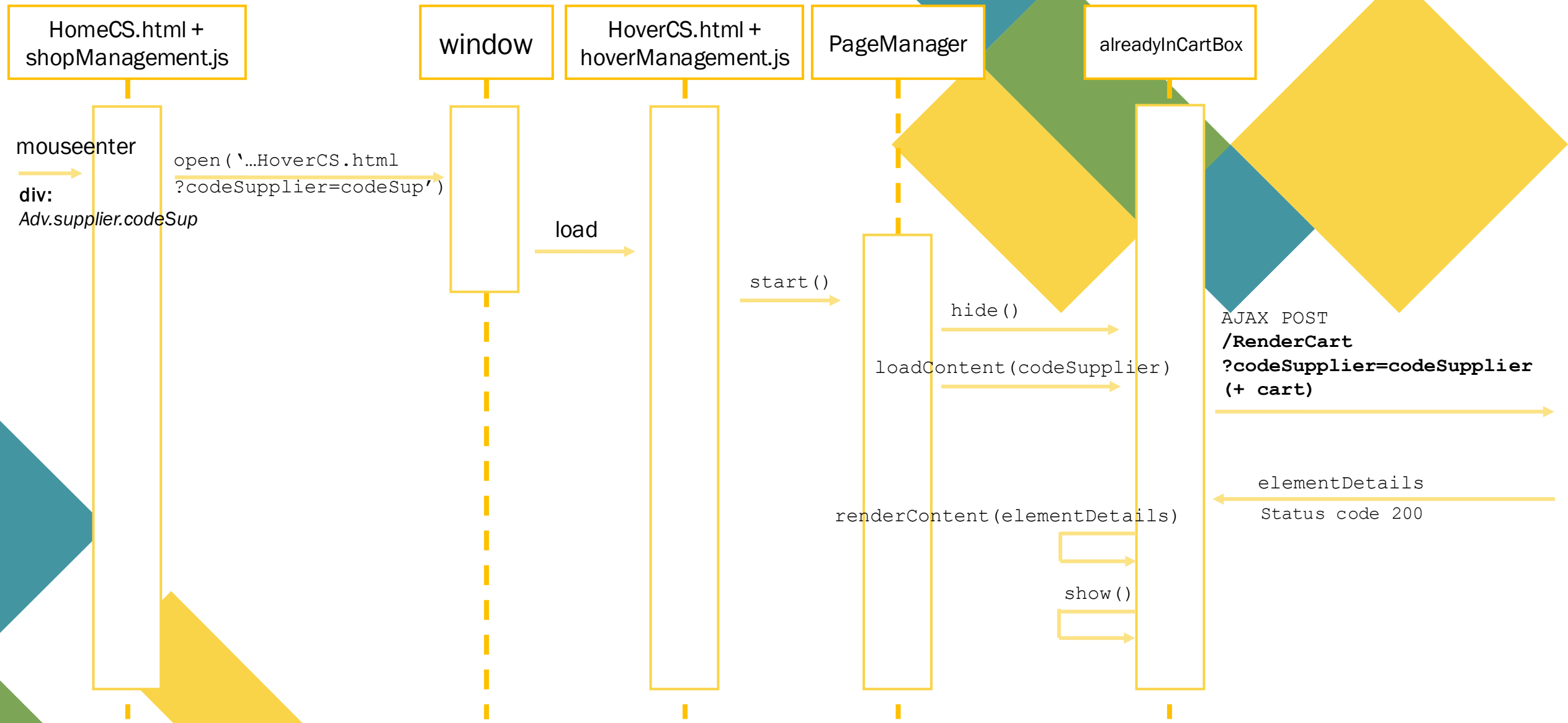


Event: Load Advertisements - *AJAX CALL*

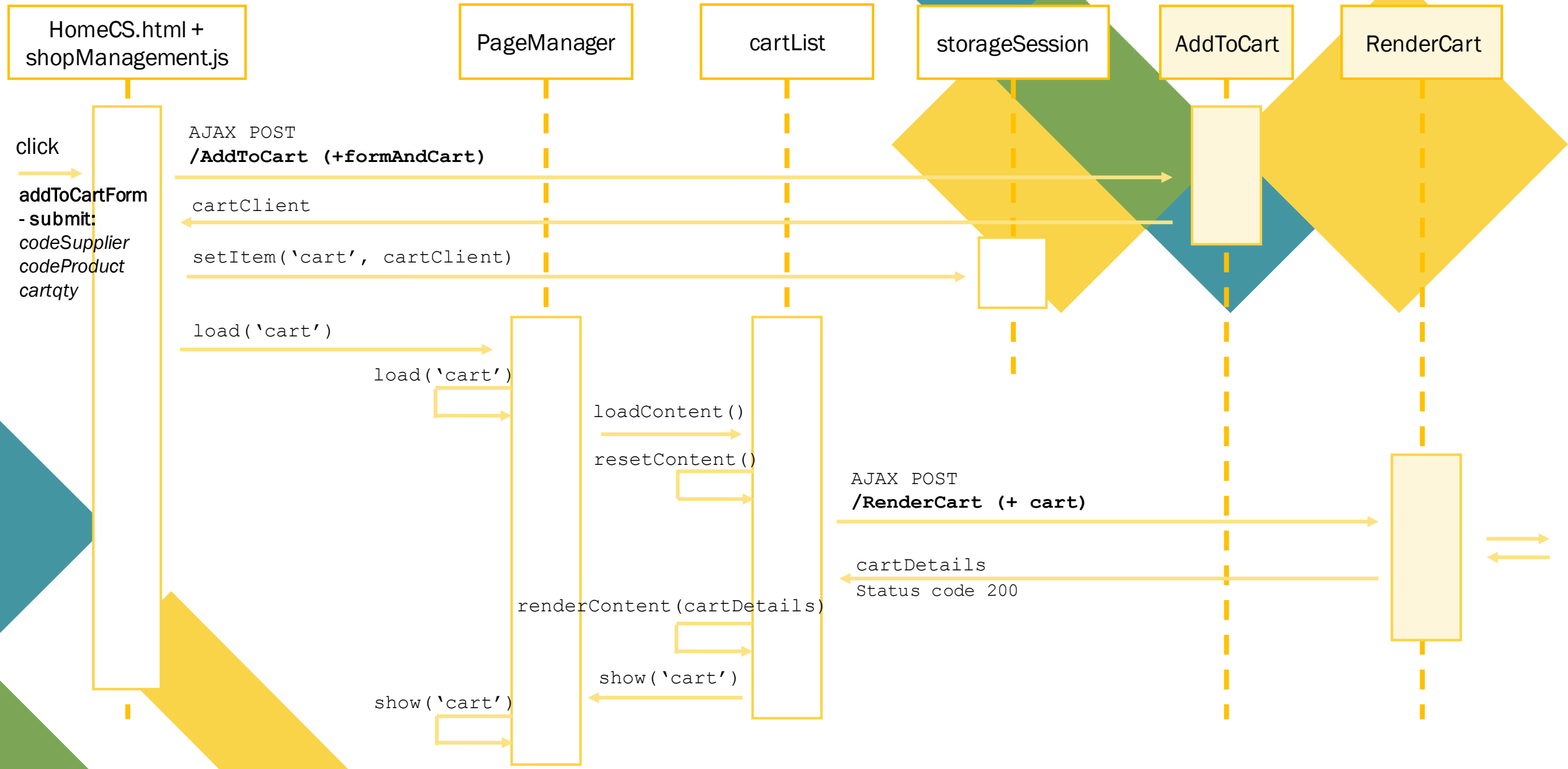
L'evento viene triggerato tramite **click** sulle ancore della lastViewedList o sulle ancore della resultsList



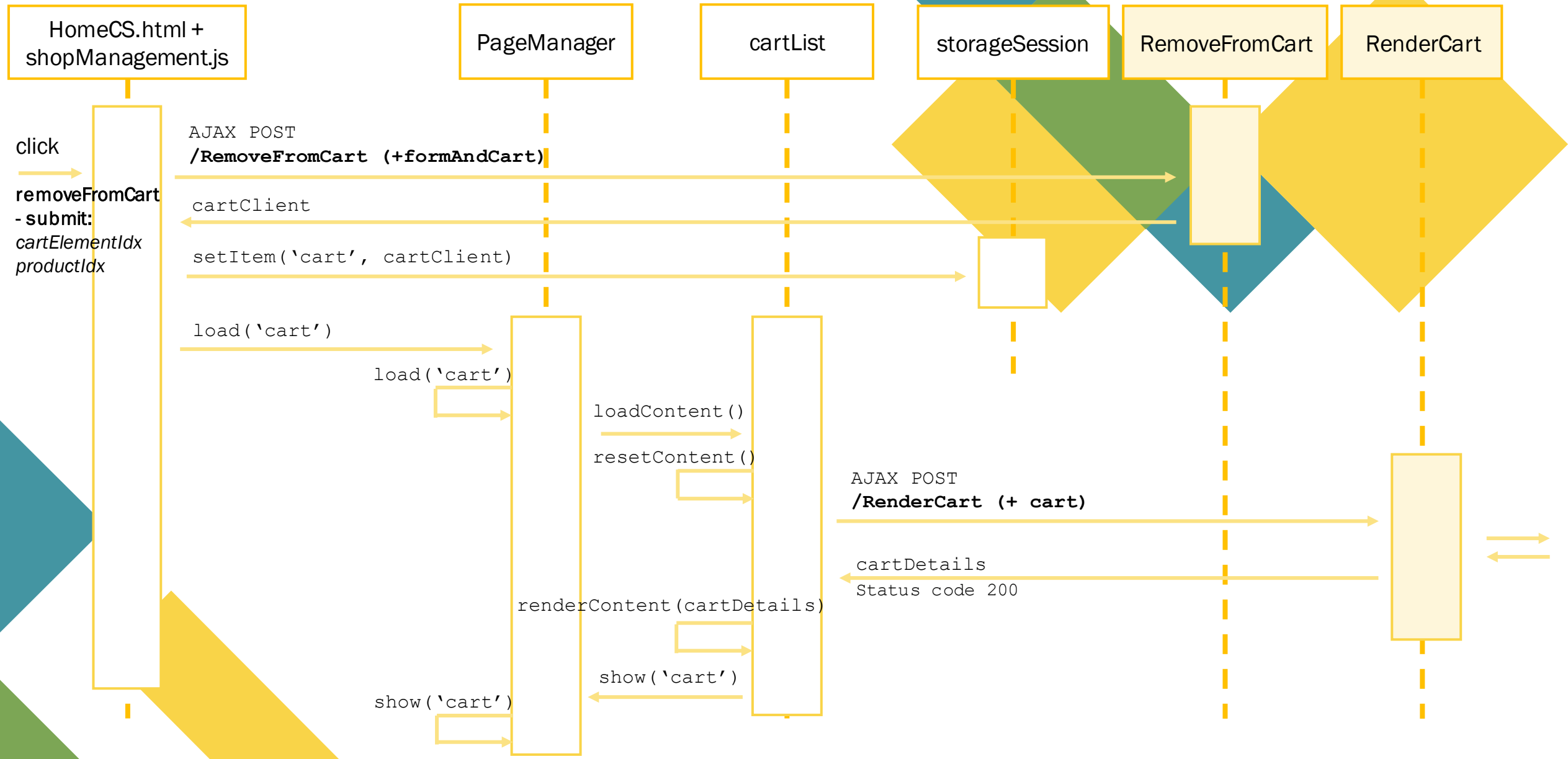
Event: Hover su AdvertisementList



Event: Add to Cart

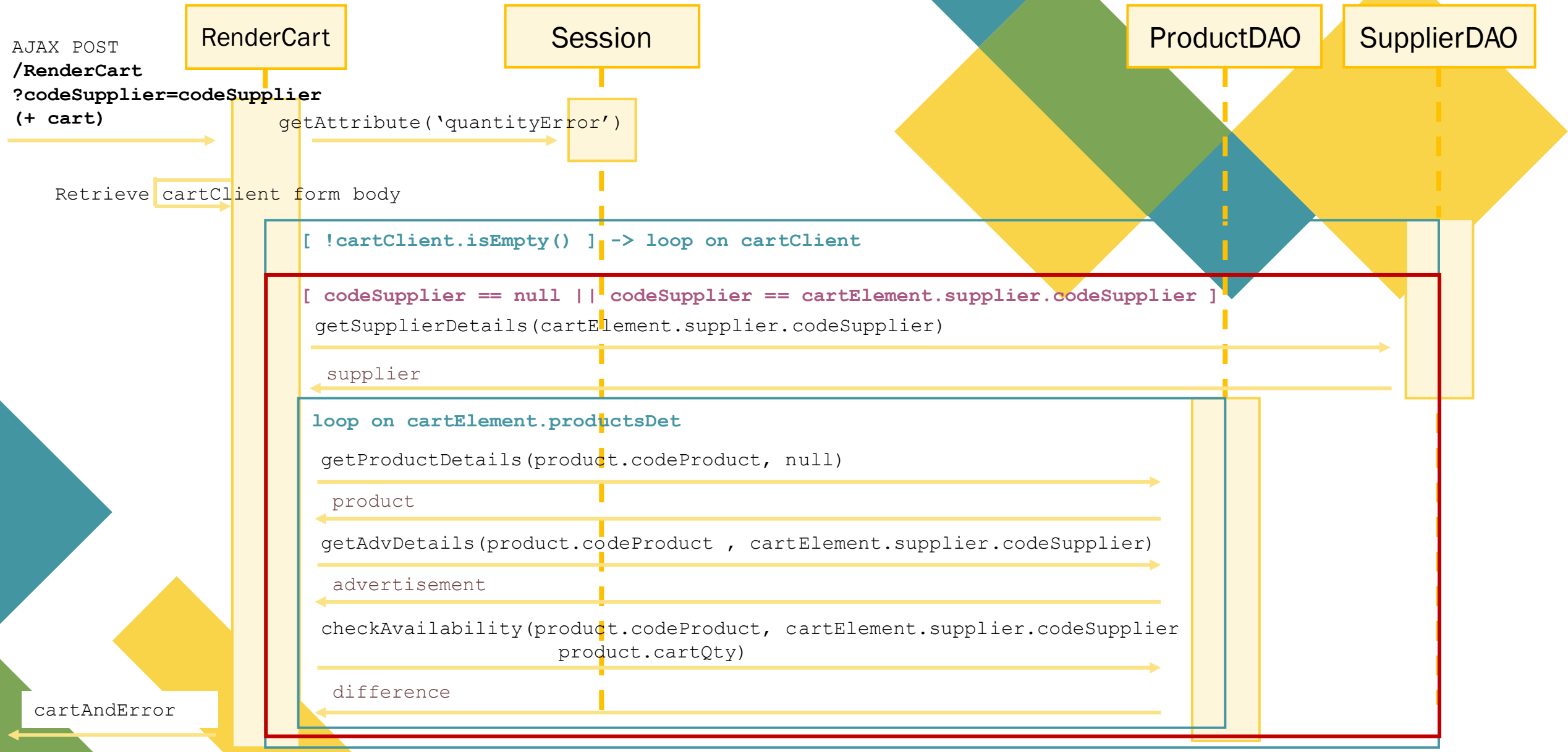


Event: Remove from Cart



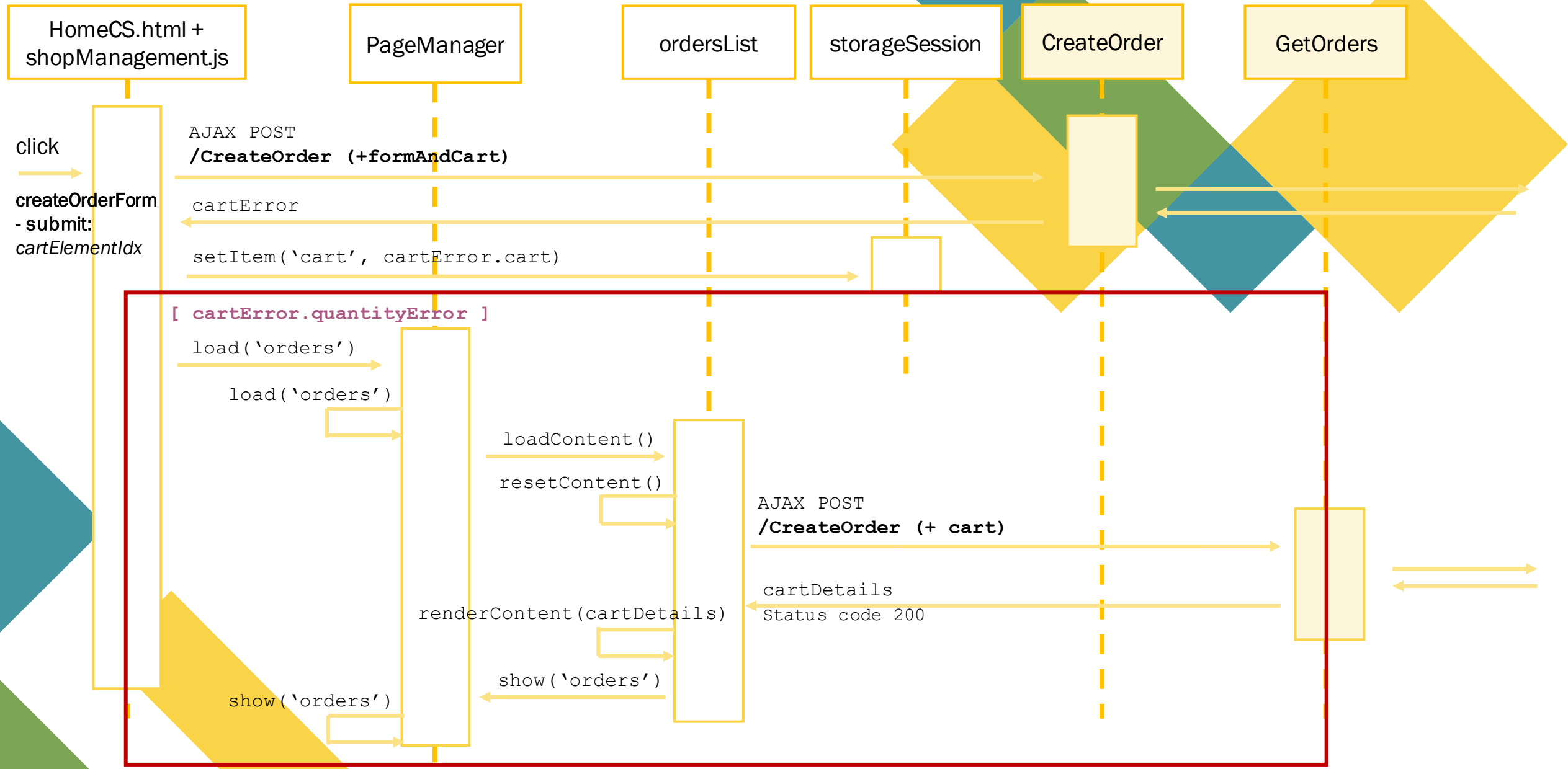
Event: Render Cart Servlet

Carica tutto il carrello o solo un elemento a seconda che sia codeSupplier sia **nullo** o meno



Event: Create Order

Ramo cartError.quantityError == true -> load('cart') omesso per evitare ripetizioni



Event: Logout