Diario di lavoro

Luogo	SAMT
Data	01.04.2019

Lavori svolti

Questa mattina ho completato in modo più completo l'inserimento dei dati nel modal che mostra i dettagli del prodotto.
Per farlo ho creato una funzione a parte che va a eliminare i dati del prodotto dal modal di dettaglio, e viene richiamata appena si chiude, in qualsiasi modo, sia premendo sulla "x" sia cliccando fuori dal modal stesso.
La funzione è quella mostrata qui sotto, che prende i dati e li elimina.

```
40  function deleteDetails(){
41
42      //Prendo tutti gli elementi dello shop  li elimino
43      var shops =  document.getElementsByClassName( classNames: "shop");
44      for (var i = 0; i < shops.length; i++){
45          shops[i].parentNode.removeChild(shops[i]);
46      }
47
48      //Svuoto la categoria del prodotto
49      document.getElementById( elementId: "category").innerHTML = "";
50
51      //Svuoto la categoria del prodotto
52      document.getElementById( elementId: "title").innerHTML = "";
53
54      //Svuoto la categoria del prodotto
55      document.getElementById( elementId: "price").innerHTML = "";
56
57      //Svuoto la categoria del prodotto
58      document.getElementById( elementId: "quantity").innerHTML = "";
59  }
```

Figura 1 funzione deleteDetails per eliminare i dettagli dal modal

Dopo questo ho lavorato al select che mostra i luoghi di ritiro nel carrello, per poterli scegliere e selezionare da dove andare a ritirare i prodotti prima di fare il check out.
Per farlo ho inserito il select, come nelle pagine dei veditori, e dopodiché una funzione JavaScript che prende dal database i dati dei luoghi e li inserisce all'interno, come mostrato dall'immagine che segue.

```
454 function setPickupShop(){
455     xhttp.onreadystatechange = function () {
456         if (this.readyState === 4 && this.status === 200) {
457
458             //Prendo i valori passati dal server e li metto in un array
459             var obj = JSON.parse(xhttp.responseText);
460
461             //Prendo la lista in cui inserire i dati
462             var ul = document.getElementsByClassName( classNames: "list")[0];
463
464             //Inserisco tutte le categorie in option e poi nel select.
465             for(var i = 0; i < obj.length; i++){
466                 var li = document.createElement( tagName: "li");
467                 li.setAttribute( qualifiedName: "class",  value: "option focus");
468                 li.setAttribute( qualifiedName: "style",  value: "z-index:0;");
469                 li.setAttribute( qualifiedName: "data-value", obj[i]["nome"]);
470                 li.appendChild(document.createTextNode(obj[i]["nome"]));
471                 ul.appendChild(li);
472             }
473         }
474     }
475     xhttp.open( method: "POST",  url: "/gestionevendita2018/customer/getPickupShop",  async: true);
476     xhttp.send();
477 }
```

Figura 2 JS setPickupShop inserisce i dati del luogo di ritiro

La funzione controller è molto simile ad altre già create, perché si occupa solo di prendere i dati richiamando la funzione apposita nel model.

```
115     public function getPickupShop()
116     {
117         //Se la sessione è aperta apro le pagine altrimeenti no
118         if(isset($_SESSION['customer'])) {
119
120             //Prendo la classe model
121             require_once 'application/models/pickupshop.php';
122             $pickupShop = new PickupShopModel();
123
124             //Array che conterrà tutti i prodotti
125             $dataArray = array();
126
127             //Prendo i dati del carrello
128             $shops = $pickupShop->getData();
129
130             //Per ogni dato inserisco il prodotto relativo nell'array
131             foreach ($shops as $value)
132                 array_push( &array: $dataArray, $value);
133
134             //Stampo con json l'array con i prodotti
135             header( string: 'Content-Type: application/json');
136             echo json_encode($dataArray);
137             //Altrimenti
138         }else{
139
140         }
141     }
```

Figura 3 controller getPickupShop richiede i dati del luogo di ritiro e lì da

Per prendere i dati dal database, ho dovuto creare una nuova classe, perché faccio riferimento una tabella, “luogo_ritiro”, a cui non avevo mai acceduto prima, di conseguenza mi serviva una nuova classe, come al solito ho istanziato il costruttore che richiama la classe “connection” come tutti gli altri model.
La funzione che prende i dati dal database e li ritorna è la seguente.

```php
public function getData()
{
    //Connetto al database
    $conn = $this->connection->sqlConnection();

    //Prendo i dati dell'utente in base alla mail
    $sql = $conn->prepare("SELECT * FROM luogo_ritiro");

    //Eseguo la query
    $sql->execute();

    $dataArray = array();
    //Se ci sono dei valori
    if ($sql->rowCount() > 1) {

        // Ciclo tutti i valori
        while ($row = $sql->fetch()) {

            array_push( &array: $dataArray, $row);
        }
    } else if ($sql->rowCount() == 1) {
        array_push( &array: $dataArray, $sql->fetch());
    }
    $conn = null;
    return $dataArray;
}
```

Figura 4 model getData prende i dati del luogo di ritiro dal database

Questo è tutto quello che ho fatto questa mattina di progetto.

Problemi riscontrati e soluzioni adottate
Non ho avuto particolari problemi.

Punto della situazione rispetto alla pianificazione
Devo completare la pagina del carrello, ovvero il checkout e la grandezza delle immagini. Devo ancora gestire il caso in cui 2 negozi facciano riferimento allo stesso prodotto e in uno dei 2 diminuisca la quantità.

Programma di massima per la prossima giornata di lavoro
Completare pagina dei clienti, implementando il checkout.