Diario di lavoro

Luogo	SAMT
Data	18.03.2019

Lavori svolti

La prima cosa che ho fatto è stata completare la mostra delle informazioni del carrello appena l'utente accede o quando inserisce un nuovo prodotto nel carrello.
Per fare questo, in JavaScript ho creato delle funzioni che se ne occupano.

```
371  function getCart(){
372      xhttp.onreadystatechange = function () {
373          if (this.readyState === 4 && this.status === 200) {
374
375              //Prendo i valori passati dal server e li metto in un array
376              var obj = JSON.parse(xhttp.responseText);
377
378              //variabile che contiene tutte le chiavi
379              var codedKeys = new Array();
380
381              if(typeof (obj[0]) == "object"){
382                  for(var i = 0; i < obj.length; i++){
383                      //richiamo la funzione che inserisce i prodotti
384                      modifyCart(obj[i]);
385                  }
386              }else{
387                  //richiamo la funzione che inserisce i prodotti
388                  modifyCart(obj);
389              }
390          }
391      }
392      xhttp.open( method: "POST",  url: "/gestionevendita2018/customer/getCart",  async: true);
393      xhttp.send();
394  }
```

Figura 1 Funzione getCart

La funzione che prende tutti i dati del carrello ha subito delle modifiche, non va più a codificare i dati ma passa direttamente i dati alla funzione che li inserisce nella pagina, questo perché la funzione "getCart" del controller passa direttamente i dati dei prodotti e non più quelli del carrello, come mostrato nell'immagine qui sotto.

```php
public function getCart()
{
    //Se la sessione è aperta apro le pagine altrimeenti no
    if(isset($_SESSION['customer'])) {

        //Prendo la classe model
        require_once 'application/models/buy.php';
        require_once 'application/models/product.php';
        $buy = new BuyModel();
        $product = new ProductModel();

        //Array che conterrà tutti i prodotti
        $prodArray = array();

        //Prendo i dati del carrello
        $cartProducts = $buy->getDataByMail($_SESSION['customer']);

        //Per ogni dato inserisco il prodotto relativo nell'array
        foreach ($cartProducts as $value)
            array_push( &array: $prodArray,
                $product->getProduct($value['nome_prodotto'], $value['prezzo_prodotto'], $value['quantita_prodotto'])[0]);

        //Stampo con json l'array con i prodotti
        header( string: 'Content-Type: application/json');
        echo json_encode($prodArray);
    //Altrimenti
    }else{

    }
}
```

Figura 2 funzione controller getCart

La funzione prende i dati dal carrello e dopo con quelli prende tutti i prodotti che l'utente ha. Oltre a questo anche quando viene aggiunto un prodotto al carrello viene richiamata un'altra funzione.

```php
function addToCart(obj){
    //Divido la chiave composta nei valori
    var keys = decode(obj.name);

    //Se il prodotto è disponibile
    if(keys[2] > 0) {
        xhttp.onreadystatechange = function () {
            if (this.readyState === 4 && this.status === 200) {

                //Diminuisco di uno il valore della quantità del prodotto nel nome
                keys[2]--;
                obj.name = encode(keys[0], keys[1], keys[2]);

                //Prendo i valori che vanno mostrati nel carrello
                getCartProduct(keys[0], keys[1], keys[2]);
            }
        }
        xhttp.open( method: "POST", url: "/gestionevendita2018/customer/addToCart", async: true);
        xhttp.setRequestHeader( name: "Content-Type", value: "application/x-www-form-urlencoded");
        xhttp.send( body: "&name=" + keys[0] + "&price=" + keys[1] + "&quantity=" + keys[2]);
    }
}
```

Figura 3 funzione addToCart

La funzione controlla che il prodotto abbia un valora maggiore di 0 e in quel caso va a richiamare la funzione e prende i valori del carrello con quel prodotto, una volta fatto diminuisce il valore che è scritto nella pagina sul prodotto e dopodiché richiama la funzione che prende il prodotto e lo inserisce nel carrello.

```
function getCartProduct(name, price, quantity){
    xhttp.onreadystatechange = function () {
        if (this.readyState === 4 && this.status === 200) {

            //Prendo i valori passati dal server e li metto in un array
            var obj = JSON.parse(xhttp.responseText);

            //richiamo la funzione che inserisce i prodotti
            modifyCart(obj[0][0]);
        }
    }
    xhttp.open( method: "POST", url: "/gestionevendita2018/customer/getCartProduct", async: true);
    xhttp.setRequestHeader( name: "Content-Type", value: "application/x-www-form-urlencoded");
    xhttp.send( body: "&name=" + name + "&price=" + price + "&quantity=" + quantity);
}
```

Figura 4 funzione getCartProduct JS

La funzione si occupa di prendere il prodotto in base ai dati passati e passarlo alla funzione che li inserisce nella pagina.
Oltre alla modifica della funzione “getCart”, nella classe controller “Customer” ci sono altre funzioni nuove, la funzione che inserisce i dati nella tabella “addToCart” e quella che prende il prodotto singolo da aggiungere al carrello.

```
    public function addToCart()
    {
        //Se la sessione è aperta apro le pagine altrimeenti no
        if(isset($_SESSION['customer'])) {

            //Prendo la classe model
            require_once 'application/models/buy.php';
            $buy = new BuyModel();

            //Prendo le variabili passate dal POST
            $name = isset($_POST["name"])? $_POST["name"] : null;
            $price = isset($_POST["price"])? $_POST["price"] : null;
            $quantity = isset($_POST["quantity"])? $_POST["quantity"] : null;

            //Se entrambi i campi non sono vuoti
            if ($name != null && $price != null&& $quantity != null) {
                $buy->insertData($name, $price, $quantity, $_SESSION['customer']);
            }
        //Altrimenti
        }else{

        }
    }
```

Figura 5 funzione addToCart

La funzione prende i valori della chiave del prodotto e poi inserisce le informazioni nella tabella grazie alla funzione model “insertData”, che ha anch’essa subito dei cambiamenti.

```php
public function insertData($prodName, $prodPrice, $prodQuantity, $custMail){
    //Connetto al database
    $conn = $this->connection->sqlConnection();
    $sql = null;

    $quantity = 0;
    $buyCount = $this->getDataByProduct($prodName, $prodPrice, $prodQuantity, $custMail);

    //Controllo se il prodotto è già inserito
    if(count($buyCount) > 0){

        //Setto la nuova quantità e modifico il campo nella tabella
        if(is_array($buyCount[0]))
            $quantity = $buyCount[0]['quantita_richiesta'] + 1;
        else
            $quantity = $buyCount['quantita_richiesta'] + 1;

        $sql = $conn->prepare("UPDATE compra SET quantita_richiesta = :quantity, data = now() WHERE nome_prodotto LIKE :prodName
        AND prezzo_prodotto LIKE :prodPrice AND
        quantita_prodotto LIKE :prodQuantity AND email_cliente LIKE :custMail");
        $sql->bindParam(':prodName', $prodName, PDO::PARAM_STR);
        $sql->bindParam(':prodPrice', $prodPrice);
        $sql->bindParam(':prodQuantity', $prodQuantity, PDO::PARAM_INT);
        $sql->bindParam(':custMail', $custMail, PDO::PARAM_STR);
        $sql->bindParam(':quantity', $quantity, PDO::PARAM_INT);
    }else{
```

Figura 6 funzione model insertData parte 1

La funzione prima controlla se il dato esiste già, se è il caso allora non inserisce un nuovo dato ma modifica quello esistente inserendo la nuova quantità richiesta del relativo prodotto, altrimenti inserisce i nuovi dati.

```php
    }else{

        //se il campo è nuovo inserisco la nuova riga
        $quantity = 1;
        $sql = $conn->prepare("INSERT INTO compra (nome_prodotto, prezzo_prodotto, quantita_prodotto, email_cliente, data, quantita_richiesta)
      VALUES (:prodName, :prodPrice, :prodQuantity, :custMail, now(), :quantity)");
        $sql->bindParam(':prodName', $prodName, PDO::PARAM_STR);
        $sql->bindParam(':prodPrice', $prodPrice);
        $sql->bindParam(':prodQuantity', $prodQuantity, PDO::PARAM_INT);
        $sql->bindParam(':custMail', $custMail, PDO::PARAM_STR);
        $sql->bindParam(':quantity', $quantity, PDO::PARAM_INT);
    }

    //Eseguo la query
    $sql->execute();
    print_r($sql->errorInfo()[2]);

    //Imposto la nuova quantità del prodotto
    require_once 'application/models/product.php';
    $product = new ProductModel();
    $product->setQuantity($prodName, $prodPrice, $prodQuantity, $prodQuantity - 1);

    //Se ci sono dei valori
    if($sql->rowCount() > 0){
        $conn = null;
        return true;
    } else {
        $conn = null;
        return false;
    }
}
```

Figura 7 funzione model insertData parte 2

Oltre alla funzione che inserisce o modifica i dati ce n'è una che prende il prodotto richiesto da inserire nel carrello e lo stampa, in modo che grazie a JavaScript si possa inserire nella pagina, come mostrato nell'immagine che segue.

```php
public function getCartProduct()
{
    //Se la sessione è aperta apro le pagine altrimeenti no
    if(isset($_SESSION['customer'])) {

        //Prendo la classe model
        require_once 'application/models/product.php';
        $product = new ProductModel();

        //Prendo le variabili passate dal POST
        $name = isset($_POST["name"])? $_POST["name"] : null;
        $price = isset($_POST["price"])? $_POST["price"] : null;
        $quantity = isset($_POST["quantity"])? $_POST["quantity"] : null;

        //Array che conterrà tutti i prodotti
        $prodArray = array();

        //Per ogni dato inserisco il prodotto relativo nell'array
        array_push( &array: $prodArray, $product->getProduct($name, $price, $quantity));

        //Stampo con json l'array con i prodotti
        header( string: 'Content-Type: application/json');
        echo json_encode($prodArray);
        //Altrimenti
    }else{

    }
}
```

Figura 8 funzione getCartProduct

Per inserire i dati, utilizzo la funzione “modifyCart” mostrata nel diario precedente a questo, ma con delle modifiche, perché oltre a mostrare i dati deve anche mostrare il numero di oggetti presenti all’interno del carrello, e se un oggetto è già presente non deve reinserirlo ma semplicemente modificare il numero di prodotti di quel tipo richiesti, come mostrato nell’immagine.

```
401     function modifyCart(obj){
402
403         //Prendo il corpo del div in cui inserire i campi
404         var divBody = document.getElementById( elementId: 'cart-list');
405
406         //Controllo se il prodotto è già nel carrello
407         if(!document.getElementById( elementId: "cart"+ obj['nome_prodotto'])) {
408
409             //Modifico la variabile del numero di oggetti
410             cartObjects += parseInt(obj['quantita_richiesta']);
411
412             //Prendo i div principale per i prodotti
413             var divContainer = document.createElement( tagName: "div");
414             divContainer.setAttribute( qualifiedName: "class",  value: "single-cart-item");
415             divContainer.setAttribute( qualifiedName: "id",  value: "cart" + obj['nome_prodotto']);
416
417             //Inserisco il link che conterrà tutto
418             var a = document.createElement( tagName: "a");
419             a.setAttribute( qualifiedName: "class",  value: "product-image");
420
421             //Inserisco l'immagine
422             var img = document.createElement( tagName: "img");
423             var src = "http://samtinfo.ch/gestionevendita2018/" + obj["img"];
424             img.setAttribute( qualifiedName: "class",  value: "cart-thumb");
425             img.setAttribute( qualifiedName: "src", src);
426             img.setAttribute( qualifiedName: "alt", obj['nome_prodotto']);
427             img.setAttribute( qualifiedName: "title", obj['nome_prodotto']);
428             img.setAttribute( qualifiedName: "id",  value: "img" + obj['nome_prodotto']);
429             a.appendChild(img);
430
431             //Creo il contenitore dei testi
432             var div = document.createElement( tagName: "div");
433             div.setAttribute( qualifiedName: "class",  value: "cart-item-desc");
```

Figura 9 funzione modifica del carrello 1

Inizialmente la funzione non passa più tutti i dati ma viene richiamata ogni volta quindi l'unica cosa che fa è inserire l'oggetto che gli viene passato tramite argomento. Un primo controllo che si può vedere, viene fatto per assicurarsi che l'oggetto con quell'id non esiste già, e se è così crea l'oggetto come mostrato in quella e nella foto che segue, con le varie informazioni e Id.

```
435             //Inserisco il bottone di cancellazione
436             var span = document.createElement( tagName: "span");
437             span.setAttribute( qualifiedName: "class",  value: "product-remove");
438             var i = document.createElement( tagName: "i");
439             i.setAttribute( qualifiedName: "class",  value: "fa fa-close");
440             i.setAttribute( qualifiedName: "aria-hidden",  value: "ture");
441             span.appendChild(i);
442             div.appendChild(span);
443
444             //Inserisco la categoria
445             var span = document.createElement( tagName: "span");
446             span.setAttribute( qualifiedName: "class",  value: "badge");
447             span.appendChild(document.createTextNode(obj['nome_categoria']));
448             div.appendChild(span);
449
450             var h6 = document.createElement( tagName: "h6");
451             h6.appendChild(document.createTextNode(obj['nome_prodotto']));
452             div.appendChild(h6);
453
454             var p = document.createElement( tagName: "p");
455             var prezzo = obj['prezzo_prodotto'] + " Fr";
456             p.setAttribute( qualifiedName: "class",  value: "price");
457             p.appendChild(document.createTextNode( data: "prezzo: " + prezzo));
458             div.appendChild(p);
459
460             var p = document.createElement( tagName: "p");
461             p.setAttribute( qualifiedName: "class",  value: "price");
462             p.setAttribute( qualifiedName: "id",  value: "quantity" + obj['nome_prodotto']);
463             p.appendChild(document.createTextNode( data: "quantità: " + obj['quantita_richiesta']));
464             div.appendChild(p);
465
466             a.appendChild(div);
467             divContainer.appendChild(a);
468             divBody.appendChild(divContainer);
```

Figura 10 funzione modifica del carrello 2

```
470         }else{
471
472             //Modifico la variabile del numero di oggetti
473             cartObjects++;
474
475             //Modifico la quantità del prodotto richiesto
476             document.getElementById( elementId: "quantity"+ obj['nome_prodotto']).innerHTML = "";
477             document.getElementById( elementId: "quantity"+ obj['nome_prodotto']).appendChild(
478                 document.createTextNode( data: "quantità: " + obj['quantita_richiesta']));
479         }
480
481         //Scrivo il numero di oggetti
482         for(var i = 0; i < document.getElementsByClassName( classNames: "cartObjects").length; i++) {
483             document.getElementsByClassName( classNames: "cartObjects")[i].innerHTML = "";
484             document.getElementsByClassName( classNames: "cartObjects")[i].appendChild(document.createTextNode(cartObjects));
485         }
486     }
```

Figura 11 funzione modifica del carrello 3

Se invece l'oggetto esiste già viene aumentata la quantità di quest'ultimo e poi modificata all'interno del carrello.
Alla fine della funzione viene preso l'oggetto che mostra il carrello in alto a destra e viene scritto il numero dei prodotti inseriti.
Dopodiché ho messo a posto l'inserimento dei prodotti da parte dei venditori, facendo in modo che venga anche inserito nella tabella "vende" che, per l'appunto, un determinato negozio vende un certo prodotto.
Per faro ho inserito un nuovo select nella pagina di inserimento che permetta di scegliere quale negozio vende il determinato prodotto, come mostrato nell'immagine sottostante.

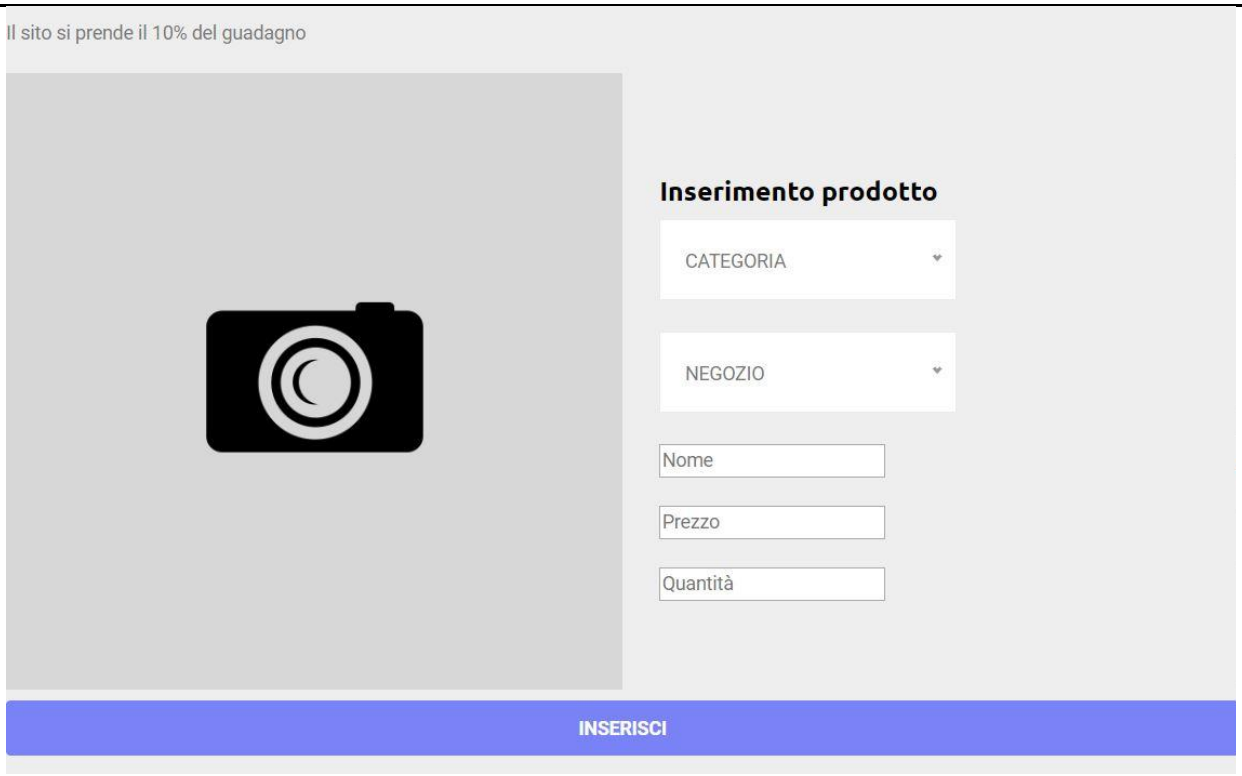

Figura 12 pagina inserimento prodotti

L'implementazione di questo select mi ha preso un po' di tempo, perché a differenza della categoria al submit non deve passare solo il dato che l'utente vede ma anche l'indirizzo e la città per poter avere la chiave completa, di conseguenza ho utilizzato anche in questo caso il metodo di concatenare in una stringa i dati e metterli come id dei vari campi, e una volta selezionato un dato viene preso l'id di questo e inserito nel select come per la categoria.
La funzione che prende i dati e li mette nl select è uguale a quella per la categoria ma fa riferimento a una tabella diversa.

```
312  function setShop(id){
313      var category = document.getElementById(id);
314
315      //Elimino l'ultimo valore se ce ne sono più di 2
316      var length = category.length;
317      if(length >= 2){
318          category.remove(1);
319      }
320      //Creo una nuova option
321      var option = document.createElement( tagName: "option");
322
323      //prendo il tag del negozio selezionato
324      var li = document.getElementsByClassName( classNames: "selected")[document.getElementsByClassName( classNames: "selected").length - 1];
325
326      //Inserisco all'interno della variabile il valore del paragrafo creato alla selezione della categoria
327      option.setAttribute( qualifiedName: "value", li.id);
328
329      //Inserisco l'option creata e la seleziono
330      category.appendChild(option);
331      category.getElementsByTagName( qualifiedName: 'option')[1].setAttribute( qualifiedName: "selected",  value: true);
332  }
```

Figura 13 funzione di modifica del select dei negozi "setShop"

La funzione che si attiva una volta selezionato un negozio però è leggermente diversa da quella della categoria, perché va a prendere il valore, non dal paragrafo con id "current" che viene creato in automatico da template ma al punto che ha la classe "select", ovvero il negozio selezionato, in modo che viene inserito nel select, non solo il nome scritto a schermo ma tutta la chiave codificata.
Una volta fatto, la funzione che inserisce i dati nel database va a inserire anche i dati nella tabella ponte che mostra quali negozi vedono quali prodotti, come mostrato nelle 2 immagini

che seguono, la prima mostra la funzione controller e la seconda il model, per il model ho creato una nuova classe che faccia riferimento alla tabella “vende”.

```php
$shop = isset($_POST["shop"])? explode( delimiter: ".", $_POST["shop"]) : null;
$sName = $shop[0];
$sAddress = $shop[1];
$sCity = $shop[2];

//Se i campi che devono comparire non sono vuoti
if ($title != null && $prize != null && $quantity != null) {
    //Inserisco i dati del prodotto
    $var = $product->insertProduct($category, $title, $prize, $quantity, $newName);

    //Se i campi che devono comparire non sono vuoti
    if ($sName != null && $sAddress != null && $sCity != null) {
        //Inserisco i dati del prodotto
        $var = $sell->insertData($sName, $sAddress, $sCity, $title, $prize, $quantity);
    }
}
```

Figura 14 inserimento prodotto con controllo negozio

```php
public function insertData($shopName, $shopAddress, $shopCity, $prodName, $prodPrice, $prodQuantity){
    //Connetto al database
    $conn = $this->connection->sqlConnection();

    //se il campo è nuovo inserisco la nuova riga
    $quantity = 1;
    $sql = $conn->prepare("INSERT INTO vende (nome_prodotto, prezzo_prodotto, quantita_prodotto, nome_negozio, ind
                           VALUES (:prodName, :prodPrice, :prodQuantity, :shopName, :shopAddress, :shopCity)");
    $sql->bindParam(':prodName', $prodName, PDO::PARAM_STR);
    $sql->bindParam(':prodPrice', $prodPrice);
    $sql->bindParam(':prodQuantity', $prodQuantity, PDO::PARAM_INT);
    $sql->bindParam(':shopName', $shopName, PDO::PARAM_STR);
    $sql->bindParam(':shopAddress', $shopAddress, PDO::PARAM_STR);
    $sql->bindParam(':shopCity', $shopCity, PDO::PARAM_STR);

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

    //Se la query va a buon fine
    if($sql->rowCount() > 0){
        $conn = null;
        return true;
    } else {
        $conn = null;
        return false;
    }
}
}
```

Figura 15 funzione insrtData classe SellModel

Una volta finito questo mi manca, oltre al completamento della pagina de venditore, come spiegato nella sezione “Punto della situazione rispetto alla pianificazione”, devo fare in modo che se lo stesso prodotto con stesso prezzo e quantità esiste in 2 negozi e uno viene comprato non venga modificato per entrambi i negozi ma venga creata una nuova istanza per il negozio a cui è stato comprato con la quantità minore.

Problemi riscontrati e soluzioni adottate

Ho messo a posto il problema riscontrato la scorsa lezione, come spiegato all’inizio della sezione precedente, e dopodiché non ho riscontrato particolari problemi.

Punto della situazione rispetto alla pianificazione

Ho completato l’inserimento nel carrello.
Ho completato l’inserimento dei prodotti, in modo che venga anche segnalato chi li vende.
Mi manca la modifica dei prodotti già inseriti.

Programma di massima per la prossima giornata di lavoro
Completare la modifica dei prodotti da parte dei venditori.