Relazione progetto Tecnologie Web.

Fabio De Palma Matricola: 833060

Tema del sito:

Il tema del progetto di Tecnologie Web è un sito di food delivery(“e-commerce”) di hamburger: Hamburger IT.

I nomi dei panini sono stati pensati proprio per i programmatori, infatti ogni nome di panino è associato ad un linguaggio di programmazione.

Dopo aver effettuato l’accesso, l’utente si trova nella schermata iniziale dove viene presentato subito il menù e può scegliere il panino da inserire nel carrello facendo un click sul nome dell’hamburger, passando sopra l’immagine di un panino viene mostrato il prezzo, il nome del panino e gli ingredienti.

Nella medesima pagina del menu sono presenti altre due sezioni:

- **Trace us:** mostra una mappa e l’indirizzo del negozio fisico.
- **About us:** mostra le varie mansioni dei dipendenti del ristorante, passando sopra con il cursore vengono mostrate le foto e i nomi.

Sotto il logo c’è una navbar che riporta alle tre sezioni: Menu, Trace us e About us.

In alto a destra, vicino al logo, è presente il nome dell’utente connesso, il bottone “disconnetti” e il carrello.

Clickando sul proprio nome, l’utente potrà vedere le proprie informazioni, modificarle e/o rimuovere completamente l’account. Il click su disconnetti chiuderà la propria sessione e reindirizzerà alla pagina di login.

In basso è presente un footer che rimanda ad un giochino di memory per non annoiare l’utente durante l’attesa del panino.

Inoltre c’è una sezione “make your hackburger” che permette all’utente di creare uno o più panini personalizzati e salvarli nella propria lista, ciò permette di prenotare più velocemente il “solito” panino. L’inserimento nel carrello avviene facendo click sul nome dell’hamburger personalizzato.

Struttura Database: hamburger

users: id(int), first_name, last_name, age, email, username, password.

name_burger: id(int), name, ingredients

Caratteristiche:

sezione login/signin:

premessa: la comunicazione con il database avviene attraverso funzioni scritte solamente nel file db.php.

ad ogni input viene applicato un quote() e alla password(signin e modifica) viene applicato una funzione di hash md5().

nella prima pagina, *index.php*, l’utente dovrà effettuare il login.

Dopo aver compilato i campi username e password, viene mandata una richiesta al server(*login.php*) attraverso ajax(*login.js*) in formato JSON(tutte le richieste ajax saranno in formato POST):

“{

“username”: “input utente”,

“password”:”password utente”

} “

Il server decodifica il JSON e verifica se l’input è valido e se l’utente è presente nel database richiamando la funzione *is_password_correct($name, $pass)* che si trova in *db.php*.

In caso di errore, viene mostrata una scritta sopra il form.

L’utente può decidere di iscriversi in caso non avesse un account.
Quindi viene rimandato in un’altra pagina *signin.php*. In questa pagina è presente un altro form che permette l’inserimento dei dati, una volta compilati tutti i campi, viene mandata una richiesta ajax dalla funzione *signin(), nel* file *signin.js,* al server *signup.php* in formato JSON.

"{

“firstname":"*name*",

"lastname":"*lastname*",

"age":"*data*",

"email":"*example@example.com*",

"username":"*username*",

"password":"*password*",

"repeatpassword":"*password*"

}"

Dal lato server viene controllato se tutti i campi sono stati inseriti, se la password rispetta un determinato pattern, ovvero l’inserimento di almeno una maiuscola, una minuscola, un numero e la lunghezza >= 8 caratteri. I controlli della password e del nome vengono effettuati utilizzando la funzione *preg_match(“@[range]@”, $string)* che ritorna 1 in caso il pattern è stato rispettato. Successivamente viene controllato, con la funzione *is_username_duplicate($username),* se quel determinato username è gia presente nel database; se così non fosse allora viene chiamata la funzione *new_user($param1,…,$paramn)* la quale inserisce tutti i dati *$param* nel database ‘users’. Dopo l’iscrizione l’utente viene reindirizzato nella pagina principale del sito *site.php.*

vengono gestite due tipologie di errore:

- In fase di login/signin: stampando un messaggio dentro un riquadro sopra i form.
- Quando l’utente è già connesso: con delle finestre modali.

Tutti gli errori vengono gestiti nel server con *die()* che ritorna un array associativo in formato json ad ajax, le ‘success’ vengono gestite in base all’array ricevuto.

Sezione make your burger: *makeit.php, check_burger.php, makeYourBurger.js*

Quando l’utente accede a questa pagina trova un panino vuoto, una lista di ingredienti a sinistra e in basso la lista dei panini salvati.

Ogni click sull’ingrediente lo fa comparire all’interno del panino con la funzione jquery .show(), aggiungendolo anche ad un array contenente tutti gli ingredienti scelti dall’utente, l’ingrediente viene poi nascosto nella sezione ingredienti a sinistra con la .hide().

Se l'utente sbaglia a inserire l'ingrediente nel panino può rimuoverlo schiacciando il relativo ingrediente dentro il panino. Quando l'utente ha scelto gli ingredienti, assegna un nome al panino e quest'ultimo viene salvato nel database 'name_burger' attraverso una chiamata ajax e passando al server(*check_burger.php*) un parametro JSON di tipo:

"{

"name": "nomepanino",

"result": "["salad","ketchup","bacon","egg"]"

}"

Quindi nel server verifico se il nome di quell'hamburger è già presente nel database controllando sull'id dell'utente, così facendo posso avere più panini con il medesimo nome per utenti diversi ma non posso avere panini con nome uguale per lo stesso utente. Il controllo viene effettuato nel database con la funzione *hamburger_alredy_exist($id, $name).*

Nella lista degli hamburger personali è presente un bottone per rimuovere l'hamburger, anche qui viene effettuata una chiamata ajax passando solamente un json con chiave "remove" e valore il "nome dell'hamburger". Quindi nel file *check_burger()* controllo se nella varibile superglobale $_REQUEST(che contiene i valori di POST) vi è scritta una chiave "remove", se così fosse, chiamo la funzione *removeBurger($id, "nome dell'hamburger")* che rimuoverà l'hamburger dal database e quindi anche dalla lista.

Sezione User: *myprofile.php, modifyinformation.js, modify.php*

In questa sezione vengono mostrate tutte le informazioni personali dell'utente dove è possibile anche effettuare la modifica (esclusa quella della data di nascita).

In *myprofile.php* utilizzo la funzione get_user_info("$_SESSION["username"]") per prendere tutte le informazioni riguardanti l'utente dal database, dopo che la pagina è stata caricata cancello tutta la sessione lasciando solo l'username dell'utente.

Quando l'utente schiaccia sul bottone "Edit Information", si apre un form dove è possibile cambiare le informazioni o rimuovere completamente l'account.

Quando l'utente vuole modificare le info, scrive all'interno dell'input text tutte le informazioni da cambiare. Quando premerà il bottone "Change data" verranno salvate tutte le informazioni in un oggetto form composto da chiave-valore simil JSON:

```
var form = {
            'first_name': $('input[name=firstname]').val(),
            'last_name': $('input[name=lastname]').val(),
            'email': $('input[name=email]').val(),
            'password': $('input[name=password]').val(),
            'repeatpassword': $("input[name=reppassword]").val(),
        };
```

questa variabile viene poi convertita in JSON e passata al server(*modify.php*) attraverso una chiamata ajax.

Dopo la decodifica del JSON, il server controlla se password e repeatpassword sono uguali, poi controlla se ci sono dei campi vuoti (in questo caso non deve avvenire la modifica), se l'utente ha immesso dei valori nel form allora vengono applicati gli stessi controlli che sono stati effettuati per la registrazione(se si rispettano tutti i pattern). Dopo tutti i controlli, viene creato un array associativo in php e inseriti tutti i dati che devono essere cambiati. Ad esempio se si vuole cambiare il nome e la mail l'array sarà così composto:

```php
$array = array(
        "first_name" => "admin",
        "email"=>"admin@gmail.com",
    );
```

in seguito parte la funzione *modify_info($array)* in *db.php.*

In questa funzione viene creata una stringa $query= “UPDATE users SET “
con un foreach che scorre tutte le chiavi dell’array $array e concatena la chiave e il valore a $query,
alla fine del ciclo viene concatenato a $query la stringa “WHERE users.username = $username”.
In questo modo la query risultante sarà:
“UPDATE users SET first_name = ‘admin’, email = ‘admin@gmail.com’ WHERE users.username = admin”

Sezione Carrello: *cart.php, cart.js, checkout.php*

In questa sezione l’utente visualizza tutti gli hamburger che ha inserito nel carrello e può modificare la quantità.

```json
var allmenuu = {
  "C": {
    "ingredients": "meat, salad, tomato, onion, pickle",
    "price":5,
    "quantity":0,
  },
  "C++":{
    "ingredients": "meat, salad, tomato, onion, pickle, cheese",
    "price":5.50,
    "quantity":0,
  },
  "JAVA": {
    "ingredients": "meat, salad, tomato, onion, pickle, bacon",
    "price":6,
    "quantity":0,
  },
  "JAVASCRIPT": {
    "ingredients": "meat, tomato, onion, pickle, cheese, bacon",
    "price":6.50,
    "quantity":0,
  },
  "PHP": {
    "ingredients": "chicken, salad, cheese, tzatziki",
    "price":5.50,
    "quantity":0,
  },
  "HTML": {
    "ingredients": "vegetables, garlic sauce, salad",
    "price":4.50,
    "quantity":0,
  },
}
```

Per la gestione e lo storage del carrello ho utilizzato la proprietà **localStorage,** questo mi permette di salvare le informazioni nel carrello e tenerle salvate per tutta la durata della sessione dell’utente. La struttura è un’oggetto javascript con chiave il nome dell’hamburger e valore un altro oggetto con ingredienti, prezzo e quantità.

Inizialmente avrà i valori del menu, poi se l’utente decide di aggiungere al carrello un proprio panino verrà inserito all’interno del localStorage. Ogni volta che devo prendere o modificare i valori nella localstorage, converto l’oggetto nella localstorage in JSON e uso le funzioni predisposte, rispettivamente getItem(chiave) e setItem(chiave, nuovi valori JSON) dove la chiave sarà sempre l’oggetto con gli hamburger.

Nella sezione cart.php è presente l’elenco dei panini inseriti nel carrello, dei bottoni per incrementare o diminuire la quantità di panini e il prezzo in base alla quantità. Ogni volta che viene fatto un click su uno dei due bottoni ”+” o “-” viene modificato il campo “quantity” della localStorage.

Sotto il carrello sono presenti due bottoni pay e reset. Reset cancella tutto il localStorage e reindirizza alla pagina principale cosicché possa ricreare l’oggetto da inserire nella localStorage con i valori standard.

Premendo pay viene chiamata la funzione total che comunica con il server(*checkout.php*) attraverso ajax passando tutto il localStorage in json. Il file *checkout.php* è utile nel caso in cui un utente cercasse di modificare il valore “price” nella localStorage nel proprio browser, in questo caso viene usato uno switch case per determinare il prezzo effettivo di ogni panino. Dopo aver fatto il controllo compare il messaggio che l’utente ha pagato la cifra corretta in base alla quantità e non la cifra modificata nel localstorage.

La funzione total viene richiamata anche per fare il totale sotto la lista dei panini cosicchè, quando viene stampato il totale, l’utente malintezionato si troverà il totale reale e non quello modificato.

Sessione Game: *game.php, game.js*

In questa pagina è presente il gioco “memory”.
Nel file *game.js* viene creato un array di oggetti “*cardObj*” che contengono il nome della carta e il path dell’immagine due volte. Dopo aver fatto click su start game viene avviato un timer e vengono mostrate le carte “girate”. Vengono usati tre array *userCard, userCardId* e *found.* Quando l’utente clicca su una carta viene mostrata l’immagine; il nome e l’id vengono salvati rispettivamente in userCard e userCardId. userCardId mi permette di verificare che non ci siano due id uguali nello stesso array (dato che gli id sono tutti diversi e univoci). Con *userCard* verifico se l’utente ha fatto click su due carte diverse e se (*userCard.length* == 2) posso verificare se i nomi delle due immagini sono uguali, così facendo azzero l’opacità con la funzione di jquery “.css()”, rimuovo l’handler click alle due carte con .off() e inserisco il nome nell’array *found,* infine se (found.length == cardObj.length/2) allora l’utente ha vinto e viene mostrato un messaggio modale.

Descrizione funzioni *db.php*

- **is_password_correct($name, $password)** -> ritorna true nel caso in cui la $password è uguale a quella presente nel database
- **ensure_logged_in()** -> viene utilizzata su ogni pagina php per determinare se l’utente ha effettuato l’accesso attraverso $_SESSION[“username”] altrimenti viene reindirizzato alla pagina di login.
- **Is_username_duplicate($username)** -> ritorna true se $username è già presente nel database.
- **new_user($firstname, $lastname,$age,$email,$username,$password)** -> inserisce il nuovo utente nel database
- **open_db()** -> ritorna un’oggetto PDO che mi permette di comunicare con il database
- **get_id()** -> ritorna l’id dell’utente
- **insert_name_of_hamburger($id, $hamname, $ingredients)** -> inserisce il nome e gli ingredienti nel database in base all’id dell’utente.
- **get_list_hamburger()** -> ritorna tutti gli elementi del database degli hamburger associati all’id
- **get_user_info($username)** -> scrive nella variabile superglobale SESSION tutti i dati dell’utente $username
- **modify_info($array)** ->modifica le informazioni dell’utente nel database in base agli elementi dentro $array
- **hamburger_alredy_exists($id, $namehamburger)** ->ritorna true se $namehamburger è già presente nel database in base all’id dell’utente.
- **remove_account($id)** ->cancella tutte le informazioni dell’utente in base all’id nella tabella users e cancella tutta la lista di panini associati a $id nella tabella name_hamburger.

Front-end:

Tutti i fogli di stile sono stati inseriti nella cartella css. hamburgerit.css è il foglio di stile che racchiude la maggior parte dello stile del sito ed è diviso in sezioni.

La sezione CARD HAMBURGER: dove ho strutturato tutte le immagini degli hamburger in griglia infatti la classe principale .row racchiude tutte le card e a questo applico un display: grid, applico le percentuali di distanziamento con grid-template-column , inoltre uso due classi: .card-front per far vedere l’immagine del panino e .card-back per i dettagli (prezzo, nome e ingredienti).

Nella sezione NAVBAR è stata inserito una regola @media cosi quando la finestra viene ridimensionata compare il menu hamburger e la navbar viene visualizzata in verticale. Al font del menu è stato poi inserito un effetto css nel file *rotationmenu.js* che alla pressione del tasto hamburger avviene un transform: rotate. La gestione delle cartelle è stata strutturata in base alle estensioni dei file. Nella cartella php è presente la cartella user dove sono stati inseriti i file che riguardano l’utente e senza alcun tag html.