



# What's In My Fridge

Dipartimento di Matematica, Informatica,  
Scienze Fisiche e Scienze della Terra  
Corso di Laurea Triennale in informatica  
Progetto di programmazione Web e Mobile  
A.A. 2021/2022

Docente: Andrea Nucita  
Studenti: Erika Casablanca, Carmelo Trifirò

# Indice

<b>1</b>	<b>What's In My Fridge</b>	<b>2</b>
1.1	Introduzione . . . . .	2
1.2	Funzionalità . . . . .	2
<b>2</b>	<b>Implementazione</b>	<b>2</b>
2.1	Il database . . . . .	3
2.2	db_config.php . . . . .	3
2.3	db_coding.php . . . . .	3
2.4	navbar.html . . . . .	4
2.5	homepage.php . . . . .	4
2.6	register.php . . . . .	4
2.7	login.php . . . . .	6
2.8	index.php . . . . .	6
2.9	myFridge.php . . . . .	7
<b>3</b>	<b>Il servizio RESTful</b>	<b>9</b>
3.1	Implementazione lato Client . . . . .	9
3.2	Implementazione lato Server . . . . .	11
<b>4</b>	<b>Possibili migliorie</b>	<b>12</b>

# 1 What's In My Fridge

## 1.1 Introduzione

What's In My Fridge (WIMF) è una web app che ha come obiettivo quello di aiutare gli utenti a gestire il proprio frigorifero, in modo da limitare gli sprechi di cibo, dovuti a possibili dimenticanze.

## 1.2 Funzionalità

WIMF permette ad ogni utente di registrarsi e di gestire il proprio frigorifero, o uno condiviso con familiari o coinquilini, il tutto con un'interfaccia semplice e intuitiva. Infatti subito dopo la registrazione o il login l'utente può creare un nuovo frigo o collegarsi a uno già esistente tramite un ID. Una volta fatto questo l'utente può accedere all'area My Fridge per iniziare a inserire i suoi alimenti all'interno del frigo tramite un semplice form dov'è possibile inserire il nome dell'alimento, la quantità o il peso e la data di scadenza. Fatto questo la web app si occuperà di inserire gli alimenti nel frigo ordinandoli per data di scadenza e offrendo un ulteriore indicatore dato dallo status.

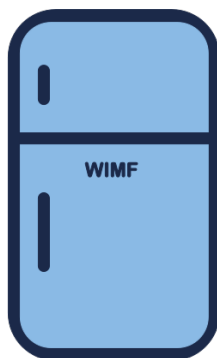


Figure 1: *Logo di What's In My Fridge*

## 2 Implementazione

Per realizzare What's In My Fridge abbiamo utilizzato vari linguaggi:

- **PHP**, utilizzato principalmente per interagire con il database tramite query in SQL;
- **HTML**, utilizzato per creare la navbar e il footer e in generale ogni pagina del sito;

- **JavaScript**, utilizzato per rendere le pagine dinamiche, sfruttato principalmente nell'area My Fridge;
- **CSS**, utilizzato per modificare il modo in cui viene visualizzata la web app;

Come servizio web abbiamo implementato un servizio RESTFUL, di cui parleremo più specificatamente in seguito.

## 2.1 Il database

Prima di parlare nello specifico del codice PHP, introduciamo il database e la sua struttura. Il database di What's In My Fridge è formato da 4 tabelle:

- **users**: id (chiave primaria auto incrementata), email, password e **id\_fridge** (chiave esterna);
- **food**: id (chiave primaria auto incrementata), **nome\_cibo**;
- **fridge**: id (chiave primaria auto incrementata);
- **contain**: **id\_frigo** (chiave esterna), **id\_cibo** (chiave esterna), quantità, grammi, **data\_scadenza**, **id\_riga** (chiave primaria auto incrementata);

## 2.2 db\_config.php

Questo file viene utilizzato come supporto per `db_coding.php`, questo infatti contiene tutte le variabili utili al collegamento al database che vengono utilizzate in ogni query.

---

```
<?php
    $dbhost="localhost";
    $dbname="what_s_in_my_fridge";
    $dbuser="admin";
    $dbpassword="";
?>
```

---

## 2.3 db\_coding.php

Questo file si occupa di effettuare tutte le possibili query utili alla web app, quindi registrazione e login dell'utente, creazione di un nuovo frigo, inserimento di un nuovo alimento, ecc...

Tutto si basa sui **PDO** (PHP Data Objects), preferiti a MySQLi in quanto possono essere utilizzati con 12 diversi sistemi di database, mentre MySQLi funziona solo con i database MySQL.

---

```
try{
    $conn = new
        PDO("mysql:host=".$GLOBALS['dbhost'].";dbname=".$GLOBALS['dbname'],
            $GLOBALS['dbuser'],$GLOBALS['dbpassword']);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
        PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT * FROM fridge WHERE id = ?");
    $stmt->execute([$id]);
    $res = $stmt->fetch();
}catch(PDOException $e){
    return array("KO", "Errore durante la creazione del frigo");
}
```

---

Come si evince da questo snippet, viene prima effettuata la connessione al database e poi viene eseguita la query con l'eventuale utilizzo di variabili, poi tramite il metodo `fetch()` estrapoliamo il risultato della query; usiamo `try` e `catch` per catturare eventuali errori durante la connessione.

## 2.4 navbar.html

Questo file viene utilizzato per creare una navbar in tutta la web app. Essa é implementata come una lista puntata ma che, tramite il CSS, appare con tutte le caratteristiche di una classica navbar. Oltre alla navbar é presente anche un footer con i nostri contatti. Questo file importa un file CSS esterno, `mystyle.css`, file che gestisce anche il CSS del body di tutte le pagine del sito.

## 2.5 homepage.php

Questa pagina é una pagina di riepilogo e presentazione con la possibilità di visitare le altre pagine tramite la navbar presentata prima. Il CSS di questa pagina é un CSS interno, che gestisce l'allineamento dei titoli, del testo e del logo della Web App.

## 2.6 register.php

In questa pagina é presente il form di registrazione dell'utente, formato da tre campi, ossia **email**, **password** e **conferma password**.

Nel momento in cui viene premuto il pulsante di registrazione viene controllato che sia stata inserita una mail valida, che sia stato inserito qualcosa nel campo password e che il campo password e conferma password siano effettivamente uguali, altrimenti riporta vari messaggi di errore.

Se é stato inserito tutto correttamente viene effettuata la registrazione tramite l'inserimento nella tabella **users** di un nuovo utente con la mail e la password, che viene criptata proprio in fase di registrazione prima di essere inserita nel database tramite la funzione:

---

```
password_hash($_POST['password'], PASSWORD_DEFAULT);
```

---

Questo metodo però funziona solo per il criptaggio, infatti non é possibile estrapolare poi la stringa corrispondente alla password ma é comunque possibile controllare in login se la password inserita é quella salvata nel database. Il CSS del form di registrazione, come anche quello del login, é scritto all'interno del file **formtype.css**, utilizzato appositamente per questo genere di form.

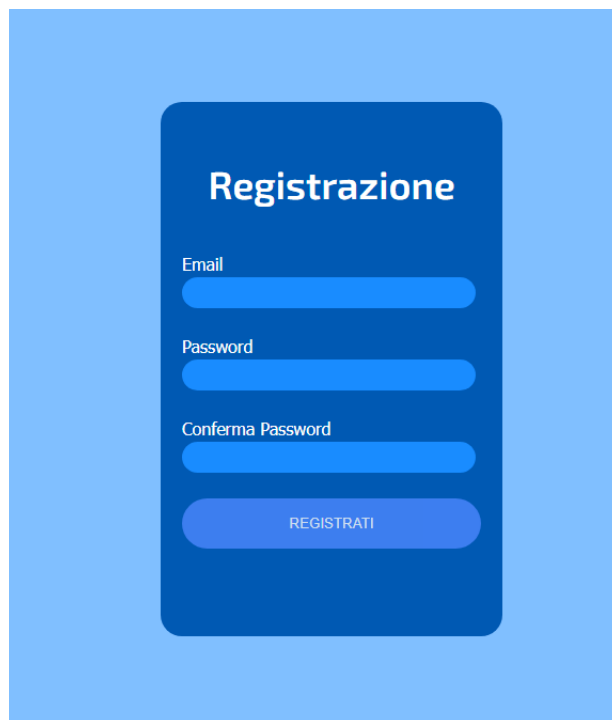
The image shows a registration form titled "Registrazione" in white text on a dark blue background. Below the title are three input fields with labels "Email", "Password", and "Conferma Password" in white text. Each field is represented by a light blue rounded rectangle. At the bottom of the form is a button labeled "REGISTRATI" in white text on a dark blue rounded rectangle. The entire form is centered on a light blue background.

Figure 2: *Form di registrazione*

## 2.7 login.php

Questa pagina gestisce invece il login dell'utente: quando questo prova ad accedere inserendo la sua mail e password, vengono effettuate delle query che controllano se esiste un utente registrato nella tabella **users** con quella mail e quella password, questa viene controllata con la funzione *password\_verify()* che prende come input la password inserita in login e la password criptata da controllare; se la password inserita ha un riscontro con quella criptata ritorna true, false altrimenti.

Fatto questo l'utente viene poi reindirizzato alla pagina **myFridge.php**. Come scritto prima, il CSS esterno di questa pagina all'interno del file **form-type.css**.

## 2.8 index.php

In questa pagina abbiamo semplicemente le impostazioni dell'utente: in questo caso si dá all'utente la possibilità di modificare la propria password (facendo un update della tabella users), effettuare il logout e di visualizzare e modificare il suo codice frigo (con un altro update sulla medesima tabella), così da poterlo condividere con i suoi familiari o coinquilini che a loro volta possono modificare il proprio codice frigo per renderlo corrispondente a quello già creato dal primo utente.



Benvenuto carmelo98.trifiro@gmail.com

Cambia password Logout

ID Frigo utente: 18

Codice frigo Ricerca

Figure 3: *Pagina impostazioni account*

## 2.9 myFridge.php

Questa pagina racchiude l'essenza della Web App: alla creazione di un nuovo account verrà chiesto all'utente di creare un nuovo frigo (inserendo un nuovo frigo nella tabella fridge) o di inserire un codice di un frigo già esistente (aggiornando la tabella users), una volta fatto sarà possibile visualizzare il proprio frigo e inserire al suo interno i vari alimenti.

Per ogni alimento inserito vanno specificati il nome (se non presente nel menù a tendina è possibile inserirlo manualmente nell'area di testo dedicata), la quantità o il peso (selezionabili tramite un radio button) e la data di scadenza. Una volta inserito l'alimento verrà inserita nella tabella contain una nuova riga contenente l'id del frigo dell'utente, l'id dell'utente, la quantità o il peso dell'alimento e la data di scadenza; poi per identificare l'alimento sfrutteremo il campo `id_row` per indicare l'indice della riga della tabella corrispondente.

Fatto questo, verrà mostrato l'alimento nell'Area My Fridge con tutte le sue informazioni, e in aggiunta verrà mostrato lo status dell'alimento e il pulsante di cancellazione.

Alimento	Quantità	Peso (grammi)	Data di scadenza	Status
plisa	3		2022-07-03	●
pomodori		500g	2022-07-03	●
zucchine		200g	2022-07-06	●
yogurt	6		2022-07-09	●
salame		100g	2022-07-09	●
uova	12		2022-07-10	●
prosciutto cotto		200g	2022-07-10	●
carne		350g	2022-07-14	●
hamburger	4		2022-07-14	●
pesche		500g	2022-07-15	●
merluzzo		150g	2022-07-15	●
speck		150g	2022-07-16	●

Figure 4: *Screenshot di un'Area My Fridge*

Lo status non è altro che un emoji di una sferetta rossa, gialla o verde, che viene mostrata accanto all'alimento in base alla sua data di scadenza:

- Rosso per alimenti scaduti o che scadono in data odierna;
- Giallo per alimenti che scadono entro una settimana dalla data odierna;
- Verde per alimenti che scadono tra più di una settimana rispetto alla data odierna;



---

```

function emoji_status(expiration_date) {
    const current_date = new Date();

    //funzione che
    //controlla la data di scadenza dell'alimento e
    //restituisce uno status con colori differenti
    const exp_date = new Date(expiration_date);

    let difference_in_time = exp_date.getTime() -
        current_date.getTime(); //variabile contenente la
    //differenza in millisecondi tra la data di scadenza e la
    //data odierna
    let difference_in_days = difference_in_time / (1000 * 3600
        * 24); //calcola invece la differenza in giorni

    if (difference_in_days < 0 )
        //se la
        //differenza minore di 0 allora l'alimento scaduto o
        //scade oggi
        return "🔴";

        //ritorna il codice corrispondente all'emoji del
        //cerchio rosso
    else if(7>difference_in_days>0)
        //se l'alimento
        //scade tra una settimana ritorna un cerchio giallo
        return "🟡";
    else if(difference_in_days>7)
        //se scade tra
        //pi di una settimana cerchio verde
        return "🟢";
}

```

---

Il tasto cancella elimina il relativo elemento dal frigo. É anche possibile aumentare o diminuire le quantità o grammi di un alimento già presente nel frigo tramite il tasto **Modifica quantità**. Bisogna però sottolineare che, affinché la modifica avvenga con successo, bisogna inserire lo stesso alimento con la stessa data di scadenza, così facendo e cliccando su **Modifica quantità** verrà modificata la quantità o il peso dell'alimento.

Il CSS relativo a questa pagina é scritto nel file **fridge.css**: questo é stato utilizzato per la visualizzazione sia del frigorifero che per tutti i "form-inline", ossia per tutte le sezioni dove vengono inseriti dei dati (eccezion fatta per

registrazione e login).

### 3 Il servizio RESTful

Abbiamo identificato come risorsa il cibo contenuto all'interno del frigo, rispettando tutti i principi architetturali dei servizi RESTful:

- Identificazione delle risorse: ogni risorsa viene identificata da un URI (Uniform Resource Identifier), nel nostro caso il cibo all'interno del frigo viene identificato da un id che rappresenta la riga in cui si trova nel database;
- Utilizzo esplicito dei metodi HTTP: ossia l'utilizzo dei metodi GET, POST, PUT e DELETE, nel nostro caso rimappati per inserire un nuovo alimento nel frigo, cancellarne uno dal frigo, modificare le quantità o grammi precedentemente inseriti o ottenere tutti gli alimenti contenuti nel frigo interessato;
- Risorse autodescrittive: nella risposta HTTP é indicato il formato dei dati inviato al client, nel nostro caso sarà del tipo text/html;
- Collegamento tra le risorse: tutto quello che un Client deve sapere su una risorsa, e sulle risorse ad essa correlate deve essere contenuto nella risposta stessa o accessibile tramite link, nel nostro caso quando per esempio andiamo ad effettuare un GET otterremo tutti i dati di tutti gli alimenti contenuti in quel frigo, quindi ID dell'alimento, ID del frigo, i grammi o la quantità, la data di scadenza e l'ID della riga corrispondente;
- Comunicazione senza stato: questa é una delle caratteristiche principali del protocollo HTTP, infatti una richiesta non ha alcuna relazione con quelle precedenti e quelle successive ad essa;

#### 3.1 Implementazione lato Client

Per implementare il servizio lato Client abbiamo creato delle funzioni che effettuano delle chiamate AJAX, utili per ottenere dati dal server senza dover ricaricare la pagina. Ogni funzione é caratterizzata dalla creazione di un oggetto XMLHttpRequest che invia una richiesta HTTP al Server, questa é composta da due parti:

- Header: che sarà composto a sua volta dal metodo, dall' URI e dal flag settato a true per indicare che la connessione stabilita deve essere asincrona;
- Body: in base all'operazione che stiamo effettuando può essere vuoto o contenere i dati da inviare (questo nel caso stesso usando come metodo POST o PUT);

Queste funzioni non hanno valori di ritorno, ma effettuano direttamente operazioni sulla pagina nel momento in cui viene eseguita la richiesta AJAX con successo tramite l'evento onload, sfruttando questo evento possiamo richiamare delle funzioni di callback per effettuare stampe, mandare alert e altro ancora.

---

```
//funzione che esegue l'operazione CRUD GET di tutti gli
    alimenti contenuti nel frigo
function showFood(callback)
{
    <?php
        $id_frigo = json_encode($_SESSION["fridge"]);
        //otteniamo l'id del frigo della sessione corrente
        echo "const fridge = ".$id_frigo. ";\n";
    ?>

    var oReq = new XMLHttpRequest();
        //apertura richiesta HTTP
    oReq.onload = function(){
        callback(oReq.responseText);           //Chiama la
            funzione di callback dandogli in input la risposta
            della richiesta HTTP appena effettuata
    };
    oReq.open("get", "api.php/contain/"+ fridge, true);
    oReq.send();
}

//funzione che esegue l'operazione CRUD DELETE di uno specifico
    alimento contenuto nel frigo e che poi aggiorner la tabella
    di tutti i cibi contenuti nel frigo

function foodDelete(id_row)
{
    var oReq = new XMLHttpRequest();
    oReq.onload = function(){
```

```

        alert('Alimento cancellato con successo!');
        //quando la richiesta viene caricata allora mostra
        l>alert con il messaggio
    showFood(printFood); //una
        volta terminata l'operazione di delete aggiorna la
        tabella chiamando la funzione showFood che a sua
        volta chiamer la funzione di callback printFood
    };
    oReq.open("delete", "api.php/contain/" + id_row, true);
        //passa l'id della riga da cancellare
        nella tabella contain
    oReq.send();
}

```

---

### 3.2 Implementazione lato Server

Nel file api.php é presente l'implementazione del servizio lato server, il codice é relativamente semplice e si occupa di creare dinamicamente una query SQL a partire dalla richiesta HTTP. Il codice é infatti in grado di identificare il metodo richiesto (GET,POST,PUT e DELETE), tramite l'URI la tabella su cui deve effettuare la query, la chiave che identifica la risorsa e il corpo della richiesta HTTP se presente.

Dopo diverse operazioni utili per creare dinamicamente il corpo di un eventuale Insert o Update, tramite uno switch andiamo a concatenare le stringhe che formeranno query da noi richiesta.

---

```

//switch che seleziona la query corrispondente all'operazione che
dobbiamo effettuare
switch ($method) {
    case 'GET':
        //se $key  valorizzata allora devi effettiare l'inner join
        perch stiamo effettuando il get degli alimenti nel
        frigo, altrimenti far un select della teballa
        $sql = "select * from ".$($key? "food inner join $table on
        food.id = $table.id_cibo WHERE id_frigo=".$key." order
        by data_scadenza" : "".$table."" )."";
        break;
    case 'PUT':
        $sql = "update '$table' SET $set WHERE id_riga=\"".$key\"";
        break;
    case 'POST':
        $sql = "insert into '$table' set $set";

```

```
        break;
    case 'DELETE':
        $sql = "delete from '$table' where id_riga=\"$key\"";
        break;
}
```

---

## 4 Possibili migliorie

Tra le possibili funzionalità sarebbe possibile inserire:

- Un registro delle attività con le modifiche effettuate sul frigo con data e ora delle modifiche;
- Implementazione di un sistema gerarchico per la gestione del frigo: il creatore di un nuovo frigo verrà identificato come master, gli altri utenti che vorranno condividere il frigo con lui manderanno una richiesta per avere un permesso. Pensato per evitare che gli utenti accedano ai frighi di altri utenti e cancellino alimenti.
- Un tasto oltre a quello della cancellazione dell'alimento che rimandi a una ricerca su Google con eventuali ricette con l'alimento corrispondente;
- La possibilità di ricevere notifiche quando un alimento é prossimo alla scadenza;