Relazione Progetto TWEB

Stefano Cipolletta - 948650

Giugno 2023

Contents

1 Tema del Sito **2**

2 Sezioni Principali **2**
2.1 Homepage . . . 2
2.2 Profilo . . . 2
2.3 Nuova Ricetta . . . 2
2.4 Nuovo Ingrediente . . . 2
2.5 Dashboard . . . 2
2.6 Pannello di Controllo . . . 3

3 Funzionalità **3**
3.1 Login/Logout . . . 3
3.2 Registrazione . . . 3
3.3 Gestione contenuto generato dall'utente . . . 3

4 Caratteristiche **4**
4.1 Usabilità . . . 4
4.2 Interazione e/o animazione . . . 4
4.3 Sessioni . . . 4
4.4 Interrogazione DB . . . 5
4.5 Validazione dati in input . . . 6
4.6 Sicurezza . . . 6

5 Front-End **6**
5.1 Separazione presentazione/contenuto/comportamento . . . 6
5.2 Soluzioni cross platform . . . 6
5.3 Organizzazione file e cartelle . . . 7

6 Back-End **8**
6.1 Schema DB . . . 8
6.2 Documentazione API . . . 8
6.2.1 Creazione Ricetta . . . 8
6.2.2 Eliminazione Ricetta . . . 9
6.2.3 Creazione Sessione . . . 9
6.2.4 Ottenere le Ricette con uno specifico Ingrediente . . . 9

1 Tema del Sito

EASY CHEF è un sito per lo scambio di ricette da cucina, l'obiettivo del sito è quello di poter condividere le proprie ricette in maniera semplice e intuitiva, anche ai meno esperti del settore.
Sono presenti due tipologie di utenti:

- *USER*: Un utente di tipo *USER* è in grado di generare contenuto all'interno del sito, quale una nuova ricetta o un nuovo ingrediente;
- *ADMIN*: Un utente di tipo *ADMIN* è in grado di vedere, aggiungere o rimuovere tutte le informazioni relative al sito (ricette, ingredienti, metodi di cottura e utenti di tipo *USER*)

2 Sezioni Principali

Il sito si presenta in un totale di 6 macro-aree;

- **Homepage** e **Profilo** sono in comune tra *USER* e *ADMIN*
- **Nuova Ricetta** e **Nuovo Ingrediente** sono accessibili solo da *USER*
- **Dashboard** e **Pannello di Controllo** sono accessibili solo da *ADMIN*

2.1 Homepage

È la pagina in cui è possibile visualizzare tutte le ricette create.
È possibile filtrare il contenuto in base agli ingredienti, al metodo di cottura e tramite una barra di ricerca presente nella navigazione.

2.2 Profilo

È la pagina in cui sono presenti le informazioni dell'*USER*/*ADMIN*.
Qui si trovano informazioni come: nome, indirizzo email, ricette create (solo per *USER*) e ricette a cui è stato messo like.

2.3 Nuova Ricetta

È la pagina utilizzata dagli *USER* per creare una nuova ricetta. Sono presenti una serie di campi da completare (es. nome ricetta, tempo di cottura, categoria, ecc...) che portano alla creazione della ricetta.

2.4 Nuovo Ingrediente

È la pagina utilizzata dagli *USER* per creare un nuovo ingrediente. È presente un campo da completare (nome dell'ingrediente) e una lista di ingredienti già presenti.

2.5 Dashboard

È la pagina utilizzata dagli *ADMIN* per vedere alcune statistiche riguardanti il servizio. Sono presenti 3 grafici raffiguranti:

- Numero di ricette per chef
- Numero di like per chef
- Numero di ricette per ingrediente

cosicchè un *ADMIN* possa farsi un'idea sull'andamento del servizio.

2.6 Pannello di Controllo

È la pagina utilizzata dagli *ADMIN* per gestire le informazioni generali del servizio. Sono presenti 4 tabelle raffiguranti:

- Utenti (‘Chefs‘)
- Ricette (‘Recipes‘)
- Ingredienti (‘Ingredients‘)
- Metodi di Cottura (‘Cooking Methods‘)

dove gli *ADMIN* possono rimuovere/modificare, o aggiungere (solo nella tabella ‘Cooking Methods‘), informazioni.

3 Funzionalità

3.1 Login/Logout

La funzionalità di **login** è suddivisa in 4 parti:

- la pagina ‘**login.php**‘ con il contenuto in html (e il relativo stile ‘auth.css‘);
- la funzione `getChefByEmail($email)` presente in ‘**chefs.php**‘ per prendere le informazioni dello chef data la sua email;
- l’endpoint **api/auth/user.php** per controllare che l’indirizzo email esista già, per la creazione della sessione e per la chiamata alla funzione `getChefByEmail($email)`;
- il controller ‘**auth.js**‘ per la validazione client-side, la chiamata AJAX e la gestione della risposta.

La funzionalità di **logout** è suddivisa in 2 parti:

- l’endpoint **api/auth/logout.php** per la rimozione della sessione;
- il controller ‘**auth.js**‘ per la chiamata AJAX e la gestione della risposta.

3.2 Registrazione

La funzionalità di **registrazione** è suddivisa in 4 parti:

- la pagina ‘**sign_up.php**‘ con il contenuto in html (e il relativo stile ‘auth.css‘);
- la funzione `setChef($name, $email, $passowrd)` presente in ‘**chefs.php**‘ per inserire in DB il nuovo utente;
- l’endpoint **api/chefs/create.php** per la creazione della sessione e la chiamata alla funzione `setChef($name, $email, $passowrd)`;
- il controller ‘**auth.js**‘ per la validazione client-side, la chiamata AJAX e la gestione della risposta.

3.3 Gestione contenuto generato dall’utente

Un utente di tipo *USER* può creare ricette e ingredienti mediante chiamate AJAX agli endpoint **api/recipes/create.php** e **api/ingredients/create.php** rispettivamente, dove verrano effettuati gli opportuni controlli (es. se la ricetta/l’ingrediente è già presente, ecc...)

4 Caratteristiche

4.1 Usabilità

Per ogni azione di un utente è presente un feedback, sia in caso di successo, sia in caso di errore, tramite una mia implementazione di 'Alert': un modale presente in '**php/navbar.php**' che viene interamente gestito grazie alla classe '***Alert***' presente in '**js/alert.js**' e tramite un cambiamento reattivo della pagina (es. like ad una ricetta).
È presente una barra di navigazione in alto ad ogni pagina (eccetto quella di login e registrazione) che rimane ancorata in cima quando si fa lo scroll; per dare profondità alla pagina rispetto alla navbar, viene aggiunto un 'box-shadow' tramite jQuery quando avviene lo scroll.
Ogni sezione all'interno del sito è intuitiva e facilmente accessibile.

4.2 Interazione e/o animazione

Sono presenti diverse interazioni da parte di un utente che non richiedono il refresh della pagina per poter effettivamente vedere i cambiamenti:

- like ad una ricetta;
- ricerca di una ricetta;
- inserimento di un ingrediente;
- rimozione di un elemento in una tabella del 'Control Panel';
- modifica di un elemento in una tabella del 'Control Panel'.

Tutte sono implementate utilizzando jQuery che, dopo aver effettuato la chiamata AJAX al server per eseguire l'operazione, aspetta l'esito e, se è andata a buon fine, aggiorna i rispettivi elementi della pagina.

4.3 Sessioni

Quando viene effettuato il login, tramite una chiamata AJAX al server viene creata la sessione con le informazioni necessarie:

```php
$_SESSION["id"] = $response["id"];
$_SESSION["name"] = $response["name"];
$_SESSION["email"] = $_POST['email'];
$_SESSION["role"] = $response["role"];
```

Quando viene effettuato il logout, tramite una chiamata AJAX al server viene distrutta la sessione:

```php
if (isset($_SESSION["id"])) {
    session_unset();
    session_destroy();
    return response(200, ["user" => json_encode(null)]);
}
```

Il controllo sulla sessione utente è implementata inserendo '**session_start()**' all'interno del file '**php/top.php**' e un controllo aggiunto in ogni pagina per accertarmi che l'utente sia effettivamente loggato, altrimenti verrà reindirizzato alla pagina di login. Viceversa, se l'utente visita la pagina di login ma è gia loggato, verrà reindirizzato alla homepage.
Lo stesso procedimento avviene per controllare il ruolo dell'utente: se un utente prova ad accedere ad una pagina nella quale è richiesto un ruolo di tipo diverso, verrà reindirizzato alla homepage.

4.4 Interrogazione DB

Tutte le funzioni per la comunicazione con il database sono nella cartella '**php/dao/**', eccetto la funzione per la connessione al DB presente nel file '**php/common.php**'.

Tutti gli utenti possono:

- vedere tutti gli ingredienti `function getAllIngredients()`
- vedere tutti i metodi di cottura `function getAllCookingMethods()`
- vedere tutte le ricette `function getAllRecipes()`
- vedere tutte le ricette dato un ingrediente `function getRecipesByIngredient($ingredient)`
- vedere tutte le ricette dato un metodo di cottura `function getRecipesByCookingMethod($cooking_method)`
- vedere le ricette a cui è stato messo like `function getRecipesLikedByChefId($chef_id)`
- aggiungere/togliere like `function setLike($recipe, $chef)`

Gli *USER* possono:

- vedere le ricette create `function getRecipesByChefId($chef_id)`
- aggiungere una ricetta
  `function createRecipe(int $chef_id, string $title, string $procedure, string $category, string $cooking_method, int $portions, int $cooking_time, array $ingredients)`
- aggiungere un ingrediente `function createIngredient($ingredient)`
- rimuovere una ricetta (solo se ne si è proprietari) `function deleteRecipe($id)`

Gli *ADMIN* possono:

- vedere tutti gli utenti iscritti `function getRecipesByIngredient($ingredient)`
- vedere numero di ricette per chef `function getNumberRecipesByChef($id)`
- vedere numero di ricette per ingrediente `function getNumberRecipesByIngredient($id)`
- vedere numero di like per chef `function getNumberLikeByChef($id)`
- rimuovere un utente `function deleteChef($id)`
- rimuovere una ricetta `function deleteRecipe($id)`
- modificare/rimuovere un ingrediente
  `function updateIngredient($oldIngredient, $newIngredient)`
  `function deleteIngredient($ingredient)`
- aggiungere/modificare/rimuovere un metodo di cottura
  `function setCookingMethod($cooking_method)`
  `function updateCookingMethod($old_cooking_method, $new_cooking_method)`
  `function deleteCookingMethod($cooking_method)`

4.5 Validazione dati in input

La validazione dei dati in input viene effettuato sia lato client che lato server.
Un esempio lato client è quello della mail:

> il controllo viene effettuato tramite la seguente RegEx:
> `/^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[A-Za-z]+$/` per assicurarmi che la mail sia formattata correttamente.

Lato server invece, vado semplicemente a controllare che i parametri che arrivano alla richiesta, se ce ne sono, siano settati e che non siano vuoti.

4.6 Sicurezza

Al fine di rendere sicura l'intera infrastruttura, lato server sono state implementate funzionalità per proteggersi da attachi XSS, HTML e SQL Injection:

- alla creazione di un utente, viene calcolato l'hash della password utilizzando la funzione apposita di PHP '**hash(...)**' che successivamente verrà inserito all'interno del DB;
```php
$password = hash("sha256", $password);
```
- ad ogni endpoint che richiede dei parametri, è stata aggiunta una funzione PHP che rimuove i tag HTML e PHP dal parametro stringa;
```php
$email = strip_tags($_POST['email']);
$password = strip_tags($_POST['password']);
```
- ad ogni query SQL con dei parametri, questi ultimi vengono passati tramite la funzione PHP bindParam dell'oggetto PDOStatement;
```php
$stmt->bindParam(1, $name, PDO::PARAM_STR);
```
- al login di un utente viene utilizzata la funzione PHP `session_regenerate_id(TRUE);` che permette di rimpiazzare il Session ID corrente con uno nuovo, così da avere sempre un Session ID nuovo ad ogni login.

5 Front-End

5.1 Separazione presentazione/contenuto/comportamento

Il sito è sviluppato separando *presentazione*, *contenuto* e *comportamento*.
Tutta la presentazione viene gestita da file css esterni e dinamicamente da javascript/jQuery quando serve;
Per javascript/jQuery viene utilizzato un approccio unobtrusive, separando in file esterni tutti gli script e aggiungendo **'event listener'** dove necessario.

5.2 Soluzioni cross platform

Per fare in modo di avere un sito cross platform sono state utilizzate:

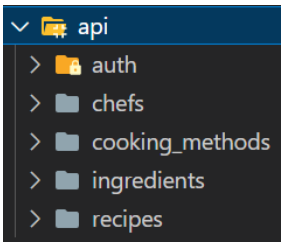
- media query nei fogli di stile
- layout flessibile (flex layout)
- layout a griglia (grid layout)

5.3 Organizzazione file e cartelle

La struttura del progetto è la seguente:

api: cartella contenente tutti gli *endpoint*;

a sua volta è suddivisa in sotto-cartelle, in modo tale da organizzare la tipologia di *endpoint* da richiamare, dove al loro interno troveremo gli effettivi *endpoint* che vengono richiamati da javascript per effettuare una determinata operazione.



---

assets: cartella contenente tutte le immagini e assets generici necessari per il sito;

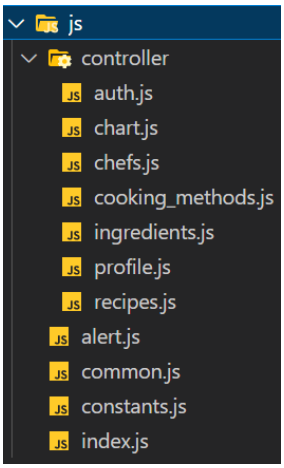
---

html: cartella contenente tutti i file in HTML in comune tra tutte le pagine;

---

js: cartella contenente tutti i file Javascript in comune e una sotto-cartella ***controller***;

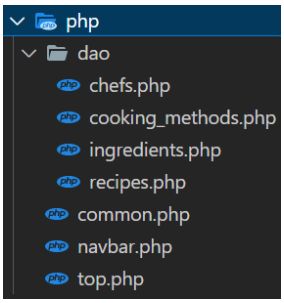
controller contiene tutti i controller Javascript per la gestione del comportamento del sito.



---

php: cartella contenente tutte le funzioni in PHP e una sotto-cartella ***dao***;

dao contiene tutte le funzioni PHP che interagiscono con il **DB**.

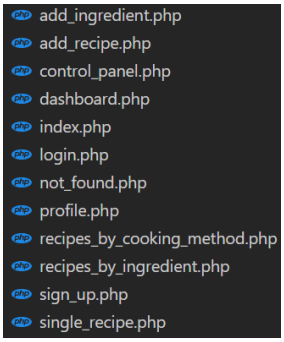


---

style: cartella contenente tutti i fogli di stile necessari per il sito;

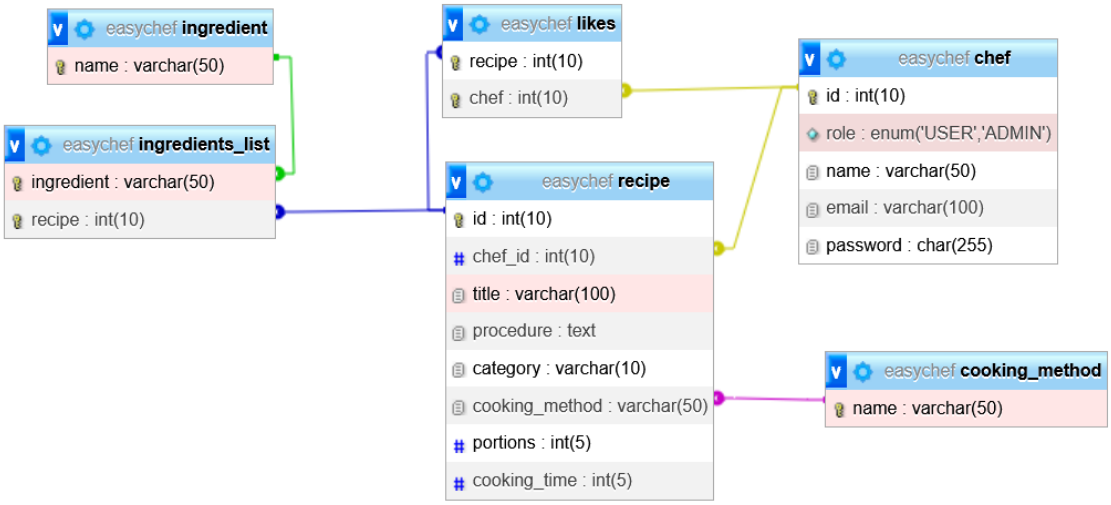
---

nella **root** della cartella del progetto sono presenti tutte le pagine necessarie al sito.



6 Back-End

6.1 Schema DB



6.2 Documentazione API

6.2.1 Creazione Ricetta

POST - *http://localhost/tweb/progetto/api/recipes/create.php*
Body Type: **json**

Body:

```
data: {
    "title": title,
    "chef_id": chef_id,
    "procedure": procedure,
    "portions": portions,
    "cooking_time": cooking_time,
    "cooking_method": cooking_method,
    "category": category,
    "ingredients": ingredients
},
```

Funzioni callback lato javascript:

```
onSuccess: (response) => {
    Alert.init(ALERT_TYPE.SUCCESS, response.ok);
    // redirect to the home page
    setTimeout(() => {
        window.location.href = "./index.php";
    }, 2000);
},
onError: (response) => {
    console.log(response);
    let error;
    if (response.responseJSON.error.toLowerCase().includes("duplicate entry")) {
        error = "Ingredient already in the list";
    } else {
        error = response.responseJSON.error;
    }

    Alert.init(ALERT_TYPE.ERROR, "Error", error);
}
```

6.2.2 Eliminazione Ricetta

POST - *http://localhost/tweb/progetto/api/recipes/delete.php*
Body Type: **json**
Body: `data: { id: recipe_id }`

Funzioni callback lato javascript:

```
onSuccess: (response) => {
    Alert.init(ALERT_TYPE.SUCCESS, "Recipe deleted", response.message);
    $(`#recipe-table #recipe-${recipe_id}`).remove();
},
onError: (response) => {
    Alert.init(ALERT_TYPE.ERROR, "An error occurred", response.responseJSON.error);
}
```

6.2.3 Creazione Sessione

POST - *http://localhost/tweb/progetto/api/auth/user.php*
Body Type: **json**
Body: `data: { email: email, password: password }`

Funzioni callback lato javascript:

```
onSuccess: (response) => {
    const user = JSON.parse(response.user);
    console.log(user);
    if (user === null) {
        Alert.init(ALERT_TYPE.SUCCESS, "You have been logged out");
        setTimeout(() => {
            window.location.href = "./login.php";
        }, 2000);
    } else {
        Alert.init(ALERT_TYPE.SUCCESS, `Welcome back ${user.name}`);
        setTimeout(() => {
            window.location.href = "./index.php";
        }, 2000);
    }
},
onError: (response) => {
    console.log("ERROR", response);
    const { error } = response.responseJSON;

    if (error.toLowerCase().includes("email")) {
        $("#login-email").css({ "border": "1px solid var(--error-color)" });
    }

    if (error.toLowerCase().includes("password")) {
        $("#login-password").css({ "border": "1px solid var(--error-color)" });
    }

    return Alert.init(ALERT_TYPE.WARNING, error);
}
```

6.2.4 Ottenere le Ricette con uno specifico Ingrediente

GET - *http://localhost/tweb/progetto/api/recipes/getByIngredient.php?**ingredient=egg***

Funzioni callback lato javascript:

```
onSuccess: (response) => {
    const recipes = JSON.parse(response.recipes);

    $(".featured-recipes h3").append(capitalize(ingredient));

    appendRecipesList(recipes, chef_id);

    $("#index-search").on("input", (e) => {
        filterRecipes(recipes, $(e.target).val().trim(), chef_id);
    });
},
onError: (response) => {
    $(".featured-recipes h3").append(capitalize(ingredient));
    $(".recipes-list").append(`<h4>${response.responseJSON.error}</h4>`);
}
```