Mensa che vorrei

Applicazione web per la gestione di una mensa universitaria

FRANCESCO MARRONE
IVAN EZZA
DANIELE SIMULA

Febbraio 2026

Indice

1 Tecnologie Web

1.1 Descrizione del sito

Con questo progetto si è voluto realizzare un sito dedicato a una mensa universitaria. È stato progettato per offrire agli studenti le funzionalità che un sito di una mensa, secondo noi, dovrebbe avere.

Il sistema consente allo studente di acquistare i pasti (i ticket per accedere alla mensa), di visualizzare i menu previsti per la settimana e la generazione di un codice QR per poter accedere fisicamente alla mensa.

È stata prevista per gli operatori una gestione lato personale della mensa di eventuali problemi nell'emissione dei pasti, nella gestione dei menu e dei piatti della mensa.

Per gli amministratori è prevista una gestione degli operatori, in quanto possono aggiungerli e rimuoverli.

1.2 Differenze tra gli utenti e credenziali account di test

Per poter realizzare le precedenti funzionalità abbiamo quindi pensato a tre ruoli distinti:

- **Utente Studente**: gli studenti possono visualizzare il loro numero di pasti, acquistare nuovi pasti e utilizzare i pasti generando un codice QR da scansionare. L'accesso avviene tramite la pagina di login, raggiungibile dalla navbar presente in ogni pagina del sito.
- **Utente Operatore**: l'operatore può visualizzare la lista degli studenti, i loro pasti e le transazioni, ed eventualmente agire modificando la quantità di pasti, modificare il menu giornaliero, aggiungere piatti ai menu e aggiungere o modificare le notizie presenti nella home. L'accesso al pannello operatore avviene tramite il percorso `/operatore/`.
- **Utente Amministratore**: l'amministratore può fare tutto quello che può fare l'operatore, ma può anche visualizzare la lista degli operatori e aggiungerne di nuovi.

Per facilitare il testing dell'applicazione, nel dump `DatabaseMensa.sql` sono già presenti degli utenti preconfigurati con le seguenti credenziali:

Ruolo	**Nome utente**	**Password**
Studente	studente	studente
Operatore	operatore	operatore
Amministratore	admin	admin

Tabella 1: Credenziali di accesso per il testing

1.3 Area Pubblica

1.3.1 Index.php

La pagina principale, così come tutte le altre pagine del sito, è strutturata utilizzando il framework CSS **Bootstrap 5** per garantire un layout responsivo e coerente.

La parte superiore della pagina presenta un *carosello* che mostra tre immagini rappresentative della mensa, con scorrimento automatico ogni 3 secondi.

Sotto il carosello viene visualizzata una griglia di notizie, recuperate dal database tramite la stored procedure `GetNotizie()`, che restituisce solo le notizie contrassegnate come attive:

```
try {
    $stmt = $conn->query("CALL GetNotizie()");
    $notizie = $stmt->fetchAll();
} catch (PDOException $e) {
    $notizie = [];
}
```

Nel caso in cui non siano presenti notizie, viene mostrato un messaggio informativo. Altrimenti, le notizie vengono disposte in card responsive (una colonna su mobile, due su tablet, tre su desktop) tramite il sistema a griglia di Bootstrap. Ogni card mostra il titolo, la descrizione, l'eventuale immagine e la data di pubblicazione.

Ciascuna card include un pulsante *"Leggi di più"* che, tramite il metodo GET, passa l'identificativo della notizia alla pagina `notizia.php`:

1.3.2 Notizia

```
<a href="notizia.php?id=<?= $notizia['id'] ?>"
   class="btn btn-primary">Leggi di piu'</a>
```

Figura 1: Codice della card notizia in "index.php"

La pagina `notizia.php` è una pagina dinamica che riceve l'identificativo della notizia tramite il metodo GET e ne carica le informazioni dal database attraverso la stored procedure `GetNotiziaById`:

Listing 1: Recupero della notizia tramite ID

```
$id = isset($_GET["id"]) ? (int) $_GET["id"] : 0;

try {
    $stmt = $conn->prepare("CALL GetNotiziaById(:id)");
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);
    $stmt->execute();
    $notizie = $stmt->fetchAll();
} catch (PDOException $e) {
    $notizie = [];
}
```

L'identificativo viene castato a intero per prevenire possibili attacchi di tipo SQL injection, e la query viene eseguita tramite un prepared statement con parametro vincolato.

Se la notizia non esiste o non è contrassegnata come attiva, viene mostrato un messaggio di errore con un pulsante per tornare alla pagina principale:

Listing 2: Controllo notizia attiva

```
<?php if (empty($notizie) || !$notizie[0]["attiva"]): ?>
    <!-- Messaggio: Notizia non trovata -->
<?php else: ?>
    <!-- Contenuto della notizia -->
<?php endif; ?>
```

Se la notizia è presente e attiva, la pagina mostra: l'immagine di copertina (se disponibile), il titolo, la descrizione, l'autore, la data di pubblicazione e il contenuto completo. I dati vengono sanitizzati tramite `htmlspecialchars()`[1] per prevenire attacchi XSS.

[1] La funzione `htmlspecialchars()` converte i caratteri speciali HTML (come `<`, `>`, `&`, `"`) nelle rispettive entità HTML, impedendo che eventuale codice malevolo inserito da un utente venga interpretato ed eseguito dal browser (attacco XSS - Cross-Site Scripting).

In fondo alla pagina sono presenti due pulsanti: uno per tornare alla pagina precedente tramite `history.back()`, e un pulsante *"Copia il link"* che utilizza l'API `navigator.clipboard` per copiare l'URL della notizia negli appunti dell'utente:

Listing 3: Funzione di condivisione del link

```
function condividi() {
    const url = window.location.href;
    navigator.clipboard.writeText(url).then(() => {
        const shareBtn = document.getElementById('shareBtn');
        const originalHTML = shareBtn.innerHTML;
        shareBtn.innerHTML = '<i class="bi bi-check-lg"></i> Link copiato!
            ';
        shareBtn.disabled = true;
        setTimeout(() => {
            shareBtn.innerHTML = originalHTML;
            shareBtn.disabled = false;
        }, 2000);
    }).catch(() => {
        alert('Errore nella copia del link');
    });
}
```

Una volta cliccato, il pulsante cambia temporaneamente il proprio testo in *"Link copiato!"* per 2 secondi, fornendo un feedback visivo all'utente.

1.3.3 Contatti

La pagina dei contatti presenta tre card, ciascuna contenente un'informazione di contatto: indirizzo, numero di telefono e indirizzo email. Ogni card include un campo di testo in sola lettura affiancato da un pulsante per copiare il contenuto negli appunti:

Listing 4: Card con input copiabile

```
<div class="input-group">
    <input type="text" class="form-control"
           value="Via Piandanna 2D" readonly>
    <button class="btn btn-outline-secondary"
            type="button" onclick="copyText(this)">
        <i class="bi bi-clipboard-fill"></i>
    </button>
```

```
</div>
```

La funzione `copyText()` utilizza l'API `navigator.clipboard`, similmente alla funzione di condivisione vista in `notizia.php`, ma in questo caso ricava il testo dal campo `input` adiacente al pulsante cliccato, risalendo all'elemento padre tramite `parentElement`:

Listing 5: Funzione di copia del testo

```
function copyText(button) {
    const input = button.parentElement.querySelector('input');
    navigator.clipboard.writeText(input.value);

    const icon = button.querySelector('i');
    icon.className = 'bi bi-clipboard-check-fill';
    setTimeout(() => {
        icon.className = 'bi bi-clipboard-fill';
    }, 1500);
}
```

Sotto le card dei contatti è presente una mappa interattiva di Google Maps, integrata tramite un `iframe`. La mappa è stata ottenuta utilizzando la funzionalità *"Incorpora mappa"* di Google Maps, che genera il codice `iframe` pronto per essere inserito nella pagina:

Listing 6: Iframe fornito da Google Maps

```
<iframe
    src="https://www.google.com/maps/embed?pb=..."
    style="width: 100%; height: 100%; border: 0;"
    allowfullscreen="" loading="lazy"
    referrerpolicy="no-referrer-when-downgrade">
</iframe>
```

1.3.4 Menù

La pagina del menu giornaliero permette di visualizzare i piatti disponibili per pranzo e cena in una determinata data. La data può essere selezionata tramite un *date picker*, il cui valore predefinito è la data odierna oppure quella eventualmente passata come parametro GET:

Listing 7: Recupero della data selezionata

```
$dataSelezionata = $_GET['data'] ?? date('Y-m-d');
```

La pagina è suddivisa in due colonne affiancate, una per il pranzo e una per la cena, che vengono popolate dinamicamente tramite una chiamata AJAX. Quando l'utente seleziona una nuova data, l'URL viene aggiornato tramite `history.pushState()` senza ricaricare la pagina, permettendo di condividere o salvare il link con la data selezionata.

get_menu.php Lo script `procedure/get_menu.php` funge da intermediario tra il frontend e il database. Riceve la data come parametro GET e restituisce i piatti del giorno in formato JSON.

Listing 8: script get_menu.php

```php
header('Content-Type: application/json');
$data = $_GET['data'] ?? date('Y-m-d');

require_once '../config/connect2DB.php';

$response = [
    'success' => true,
    'data' => ['pranzo' => [], 'cena' => []]
];

try {
    $stmt = $conn->prepare("CALL GetMenuDelGiorno(?)");
    $stmt->execute([$data]);
    $results = $stmt->fetchAll(PDO::FETCH_ASSOC);

    if (count($results) === 0) {
        echo json_encode(['success' => false,
             'error' => 'Nessun menu trovato']);
        exit;
    }

    foreach ($results as $row) {
        if ($row['tipo_pasto'] == 'Pranzo') {
            $response['data']['pranzo'][] = $row;
        } else {
            $response['data']['cena'][] = $row;
        }
    }

    echo json_encode($response);
```

```
} catch (PDOException $e) {
    echo json_encode(['success' => false,
        'error' => 'Errore nel recupero del menu']);
}
```

Lo script imposta innanzitutto l'header `Content-Type` a `application/json` per indicare al client che la risposta è in formato JSON. La data viene recuperata dalla query string tramite l'operatore *null coalescing* (`??`), che in caso il valore fosse null imposta un valore default indicato alla destra di "??", in questo caso la data di oggi.

Viene invocata la procedura `GetMenuDelGiorno`. I risultati vengono recuperati come array associativo tramite `fetchAll(PDO::FETCH_ASSOC)` e successivamente smistati in due array distinti, `pranzo` e `cena`, in base al campo `tipo_pasto` di ciascun record. In caso di assenza di dati o di errore del database, viene restituito un oggetto JSON con il campo `success` impostato a `false` e un messaggio di errore.

caricaMenu() La funzione `caricaMenu()` viene invocata sia al caricamento iniziale della pagina sia ogni volta che l'utente seleziona una nuova data:

Listing 9: Funzione caricaMenu()

```
function caricaMenu(data) {
    nascondiAlert();

    fetch('procedure/get_menu.php?data=' + data)
        .then(response => response.json())
        .then(result => {
            if (result.success) {
                aggiornaColonna('pranzo', result.data.pranzo);
                aggiornaColonna('cena', result.data.cena);
            } else {
                svuotaColonne();
                mostraAlert('warning',
                    'Nessun menu disponibile');
            }
        })
        .catch(error => {
            svuotaColonne();
            mostraAlert('danger',
                'Impossibile caricare il menu');
```

```
        });
}
```

La funzione utilizza l'API `fetch()` per effettuare una richiesta GET asincrona verso lo script `get_menu.php`, passando la data come parametro nell'URL. La risposta viene convertita da JSON a oggetto JavaScript tramite `response.json()`. Se il campo `success` è `true`, la funzione invoca `aggiornaColonna()` due volte: una per il pranzo e una per la cena, passando il rispettivo array di piatti. In caso di insuccesso o di errore nella comunicazione con il server, le colonne vengono svuotate e viene mostrato un alert di avviso all'utente.

La funzione aggiornaColonna() La funzione `aggiornaColonna()` è responsabile della traduzione dei dati JSON ricevuti dal server in elementi HTML visibili nella pagina:

Listing 10: Funzione aggiornaColonna()

```
function aggiornaColonna(tipo, piatti) {
    // Seleziona la colonna HTML corretta
    const colonna = tipo === 'pranzo' ?
        document.getElementById('colonnaPranzo') :
        document.getElementById('colonnaCena');

    // Svuota la lista esistente
    const lista = colonna.querySelector('.list-group');
    lista.innerHTML = '';

    if (piatti.length > 0) {
        piatti.forEach(piatto => {
            const li = document.createElement('li');
            li.className = 'list-group-item';

            // Costruisce l'HTML con nome del piatto
            let html = '<span>'
                + piatto.piatto_nome + '</span>';

            // Aggiunge i badge degli allergeni
            if (piatto.allergeni) {
                const allergeniArray =
                    piatto.allergeni.split(', ');
                html += '<div class="mt-1">';
                allergeniArray.forEach(allergene => {
```

```
                        html += '<span class="badge bg-warning">'
                            + allergene + '</span>';
                    });
                    html += '</div>';
                }

                // Inserisce l'HTML nel <li> e lo aggiunge
                li.innerHTML = html;
                lista.appendChild(li);
            });
        } else {
            const li = document.createElement('li');
            li.className = 'list-group-item text-muted';
            li.textContent = 'Non disponibile';
            lista.appendChild(li);
        }
    }
```

La funzione riceve come parametri il tipo di pasto (la stringa `'pranzo'` o `'cena'`) e l'array di piatti restituito dalla chiamata AJAX. Per prima cosa individua la colonna HTML corretta tramite querySelector e svuota la lista impostandol'`innerHTML` a stringa vuota.

Se l'array dei piatti non è vuoto, la funzione itera su ciascun piatto con `forEach`, creando un elemento `<li>` per ognuno. Per ogni piatto viene costruita una stringa HTML contenente il nome del piatto e, se presenti, i relativi allergeni. La stringa degli allergeni, che arriva dal database come un unico valore separato da virgole (ad esempio `"Glutine, Lattosio"`), viene suddivisa in un array tramite `split(', ')` e per ciascun allergene viene generato un badge Bootstrap di colore giallo con un'icona di avvertimento.

Infine, le due righe `li.innerHTML = html` e `lista.appendChild(li)` completano il processo: la prima inserisce la stringa HTML costruita come contenuto dell'elemento `<li>`, mentre la seconda aggiunge fisicamente quell'elemento alla lista nel DOM[2], rendendolo visibile all'utente. Se invece l'array è vuoto, viene inserito un singolo elemento con il testo *"Non disponibile"*.

[2]Il DOM (Document Object Model) è la rappresentazione ad albero della pagina HTML in memoria. Modificare il DOM tramite JavaScript permette di aggiornare dinamicamente il contenuto visibile della pagina senza ricaricarla.

1.4 Area Studenti

1.4.1 Registrazione e Login

Flusso Completo di Autenticazione Il processo di autenticazione segue questo flusso:

1. **Presentazione Form**: L'utente accede a `login.php`
2. **Validazione Client**: JavaScript verifica i dati prima dell'invio
3. **Submit**: Il form viene inviato a `accesso.php` o `registrazione.php`
4. **Validazione Server**: Controllo `empty()` su tutti i campi
5. **Query Database**: Verifica credenziali o inserimento nuovo utente
6. **Creazione Sessione**: Se successo, crea `$_SESSION['nomeUtente']` e `$_SESSION['agent']`
7. **Redirect**: Indirizzamento a `area_personale.php`
8. **Verifica Continua**: Ad ogni richiesta, verifica del session agent

Il sistema implementa un doppio meccanismo di autenticazione che gestisce sia l'accesso per utenti esistenti sia la registrazione di nuovi studenti. L'architettura si basa su un approccio stratificato di validazione e sicurezza, con particolare attenzione alla prevenzione di attacchi comuni come SQL injection, session hijacking e form tampering.

Architettura del Form Unificato Il file `login.php` implementa un form dinamico che si adatta in base al parametro GET `new`:

Listing 11: Gestione dinamica del form di login/registrazione

```
<?php
if (isset($_GET['new']) && $_GET['new'] == '1') {
    echo '<form action="registrazione.php" class="needs-validation"
          novalidate method="post">';
    echo '<h1>Inserisci i tuoi dati e registrati</h1>';
} else {
    echo '<form action="accesso.php" class="needs-validation"
          novalidate method="post">';
    echo '<h1>Inserisci i tuoi dati per accedere</h1>';
}
?>
```

Questa struttura consente di:

- Condividere la logica di validazione tra login e registrazione
- Ridurre la duplicazione del codice
- Mantenere un'interfaccia utente coerente
- Semplificare la manutenzione del codice

Validazione Client-Side con Bootstrap 5 Il sistema utilizza la classe `needs-validation` di Bootstrap 5 combinata con l'attributo HTML5 `novalidate` per implementare una validazione client-side personalizzata:

Listing 12: Script di validazione Bootstrap 5

```
(() => {
    'use strict'
    const forms = document.querySelectorAll('.needs-validation')

    Array.from(forms).forEach(form => {
        form.addEventListener('submit', event => {
            if (!form.checkValidity()) {
                event.preventDefault()       // Blocca l'invio
                event.stopPropagation()      // Ferma il bubbling
            }
            form.classList.add('was-validated')
        }, false)
    })
})()
```

Questo approccio:

- Disabilita la validazione nativa del browser con `novalidate`
- Intercetta l'evento `submit` prima dell'invio del form
- Verifica la validità degli input usando `checkValidity()`
- Blocca l'invio se i dati non sono validi
- Applica gli stili Bootstrap per feedback visivo

Regex Pattern per la Validazione Il sistema implementa una serie di pattern regex specifici per ciascun campo, garantendo l'integrità dei dati già lato client:

Tabella 2: Pattern di validazione implementati

Campo	**Pattern Regex**	**Descrizione**
Nome Utente	`^[A-Za-z0-9]{3,30}$`	Alfanumerico, 3-30 caratteri
Password	`[A-Za-z0-9]+`	Alfanumerico, minimo 3 caratteri
Email	`^[^\s@]+@[^\s@]+\.[^\s@]+$`	Formato email standard
Nome/Cognome	`^[A-Za-zÀ-ÿ\s']{3,30}$`	Lettere (anche accentate), 3-30 caratteri
Codice Fiscale	`^[A-Za-z0-9]{16}$`	16 caratteri alfanumerici
Città	`[A-Za-zàèéìòù ]{2,30}`	Lettere con accenti, 2-30 caratteri

Si è consapevoli che alcuni di questi Regex (in particolare quelli della Password) non sono sufficienti in un sistema reale ma si è deciso di optare per questi tipi di regex per comodità nel testing, annoto qua un regex password più realistico: `(?=.*[A-Z])(?=.*[0-9])(?=.*[@#!?]).+`

Listing 13: Esempio di campo con validazione regex

```
<input type="text"
       class="form-control"
       name="nomeUtente"
       id="nomeUtente"
       pattern="[A-Za-z0-9]{3,30}"
       minlength="3"
       maxlength="30"
       required>
<div class="invalid-feedback">
    Il nome utente deve essere di minimo 3 caratteri e massimo 30,
    puo contenere solo lettere e numeri
</div>
```

Validazione Server-Side Nonostante la validazione client-side, il sistema implementa controlli server-side:

Listing 14: Controlli di sicurezza server-side

```php
// Controllo di sicurezza aggiuntivo
if (empty($nomeUtente) || empty($password) || empty($email) ||
    empty($nome) || empty($cognome) || empty($sesso) ||
    empty($dataNascita) || empty($indirizzo) || empty($cf) ||
    empty($citta) || empty($isee)) {
    header("Location: login.php?error=1&new=1");
    exit();
}
```

Questo approccio protegge da:

- Rimozione dell'attributo `required` tramite DevTools
- Modifica dei pattern regex nel DOM
- Invio diretto di richieste POST bypassando il form
- Manipolazione JavaScript lato client, ecc..

Calcolo Automatico della Fascia ISEE Durante la registrazione, il sistema calcola automaticamente la fascia di appartenenza basandosi sul valore ISEE dichiarato:

Listing 15: Funzione di calcolo fascia ISEE

```php
function calcolaFascia($isee) {
    if ($isee <= 9999) {
        $fascia = 1;  // Fascia economica (2.30 EUR)
    } else if ($isee >= 10000 && $isee <= 19999) {
        $fascia = 2;  // Fascia intermedia (4.00 EUR)
    } else {
        $fascia = 3;  // Fascia standard (5.50 EUR)
    }
    return $fascia;
}
```

La fascia determinata viene poi utilizzata per:

- Inserimento nel database nella tabella `info_studente`
- Calcolo automatico del costo dei pasti

Session Agent: Protezione contro Session Hijacking Il sistema implementa un meccanismo di sicurezza avanzato basato su **session fingerprinting** utilizzando l'User-Agent del browser:

Listing 16: Implementazione del Session Agent

```php
// Creazione del fingerprint durante login/registrazione
$_SESSION['nomeUtente'] = $nomeUtente;
$_SESSION['agent'] = sha1($_SERVER['HTTP_USER_AGENT']);

// Verifica del fingerprint ad ogni richiesta
if (isset($_SESSION['agent']) &&
    ($_SESSION['agent'] == sha1($_SERVER['HTTP_USER_AGENT'])) &&
    isset($_SESSION['nomeUtente'])) {
    // Sessione valida
    header("Location: area_personale.php");
} else {
    // Sessione compromessa o invalida
    session_destroy();
    header("Location: index.php");
}
```

Funzionamento del meccanismo:

1. **Creazione**: Al momento del login/registrazione, viene calcolato l'hash SHA-1 dell'User-Agent del browser dell'utente e memorizzato in sessione
2. **Verifica**: Ad ogni richiesta successiva, il sistema ricalcola l'hash dell'User-Agent corrente e lo confronta con quello memorizzato
3. **Invalidazione**: Se gli hash non corrispondono, la sessione viene distrutta e l'utente viene reindirizzato alla home

Protezione offerta:

- **Session Fixation**: Anche se l'attaccante conosce il session ID, non può utilizzarlo da un contesto diverso

Limitazioni note: Si noti che questa è una dimostrazione di come si possono implementare dei metodi per proteggersi da tentativi di spoofing e session hijacking ma si è consapevoli che tale approccio può essere bypassato tramite User-Agent spoofing, in un contesto reale si dovrebbe implementare un fingerprinting multi-fattoriale ad esempio includendo indirizzo IP, lingua del browser e altre caratteristiche.

Gestione delle Password Il sistema utilizza le funzioni native PHP per l'hashing sicuro delle password:

Listing 17: Hashing e verifica password

```php
// Durante la registrazione
$password_hash = password_hash($password, PASSWORD_DEFAULT);

// Durante il login
if (password_verify($password, $result['passwordhash'])) {
    // Password corretta
    $_SESSION['nomeUtente'] = $nomeUtente;
    $_SESSION['agent'] = sha1($_SERVER['HTTP_USER_AGENT']);
    header("Location: area_personale.php");
}
```

`PASSWORD_DEFAULT` utilizza un algoritmo che:

- Genera salt automatico e casuale
- Cost factor di 10 ($2^{10} = 1024$ iterazioni)
- Resistenza agli attacchi rainbow table[3]

Stored Procedure per la Registrazione La registrazione utilizza una stored procedure per garantire atomicità e ridurre il rischio di SQL injection:

Listing 18: Chiamata alla stored procedure

```php
$query = "CALL regNewStudente(:nomeutente, :email, :password, :cf,
          :nome, :cognome, :sesso, :datanascita, :indirizzo,
          :citta, :fascia)";
$stmt = $conn->prepare($query);
$stmt->bindParam(':nomeutente', $nomeUtente, PDO::PARAM_STR);
$stmt->bindParam(':email', $email, PDO::PARAM_STR);
$stmt->bindParam(':password', $password_hash, PDO::PARAM_STR);
// ... altri parametri
if ($stmt->execute()) {
    $stmt->closeCursor();
    $_SESSION['nomeUtente'] = $nomeUtente;
```

[3]Una rainbow table Contiene hash precalcolati di milioni di possibili password, permettendo di trovare rapidamente la password originale dato il suo hash. L'uso di salt (valori casuali aggiunti alla password prima dell'hashing) rende inutili le rainbow table poiché ogni password, anche identica, produce hash differenti.

```
    $_SESSION['agent'] = sha1($_SERVER['HTTP_USER_AGENT']);
    header("Location: accesso.php");
}
```

L'uso di prepared statements con parametri bound previene:

- SQL Injection
- Attacchi di manipolazione del database
- Errori di tipo di dato

Gestione degli Errori e Feedback Utente Il sistema gestisce gli errori attraverso redirect con parametri GET:

Listing 19: Gestione errori e feedback

```
// In caso di errore
header("Location: login.php?error=1&new=1");

// Nel form, visualizzazione dell'errore
if (isset($_GET['error']) && $_GET['error'] == '1')
    echo '<div class="alert alert-danger">
          Credenziali non valide
          </div>';
```

Questo approccio:

- Previene il re-submit del form tramite refresh
- Mantiene l'esperienza utente fluida
- Non espone informazioni sensibili nell'errore
- Utilizza il pattern Post-Redirect-Get (PRG)

Validazione della Data di Nascita Un esempio di validazione complessa è implementato per la data di nascita:

Listing 20: Validazione dinamica data di nascita

```
// Imposta il massimo ad oggi
document.getElementById('dataNascita').max =
    new Date().toISOString().split('T')[0];

// Imposta il minimo a 100 anni fa
let minDate = new Date();
minDate.setFullYear(minDate.getFullYear() - 100);
document.getElementById('dataNascita').min =
    minDate.toISOString().split('T')[0];
```

Questo impedisce:

- Date future (nascite non ancora avvenute)
- Date troppo nel passato (oltre 100 anni)
- Input di date non realistiche

Conferma Password Per la registrazione, viene implementato un meccanismo di conferma password:

Listing 21: Funzione di conferma password

```
function confermapsw(valore) {
    if (document.getElementById("password").value != valore) {
        alert("Le password non corrispondono, riprova.");
        document.getElementById("pswConferma").value = "";
        document.getElementById("pswConferma").focus();
    }
}
```

Chiamata tramite:

```
<input type="password"
       name="pswConferma"
       id="pswConferma"
       onchange="confermapsw(this.value)">
```

Tabindex per Accessibilità Tutti gli input implementano l'attributo `tabindex` per garantire una navigazione logica tramite tastiera:

Listing 22: Esempio di gestione tabindex

```
<input type="text" name="nomeUtente" tabindex="1">
<input type="password" name="password" tabindex="2">
<input type="password" name="pswConferma" tabindex="3">
<input type="email" name="email" tabindex="4">
```

1.4.2 Area Personale

La pagina `area_personale.php` rappresenta il pannello di controllo dello studente, accessibile solo dopo aver effettuato il login. Questa pagina permette allo studente di visualizzare i pasti disponibili, acquistare nuovi pasti, generare il QR code per l'accesso alla mensa e visualizzare le proprie informazioni personali.

Sistema di Generazione del QR Code Quando lo studente clicca sul pulsante "Utilizza Buono", viene attivata una funzione JavaScript che effettua una richiesta AJAX al server:

Listing 23: Generazione del QR code

```

if (data.success) {
    document.getElementById('qrcode').innerHTML = '';
    new QRCode(document.getElementById('qrcode'), {
        text: data.token,
        width: 200,
        height: 200,
        colorDark: '#000000',
        colorLight: '#ffffff',
        correctLevel: QRCode.CorrectLevel.H
    });
```

Il codice richiama la procedura `genera_token.php` che crea nel database un token composto dalla concatenazione di tre elementi: l'identificativo dell'utente, l'identificativo del pasto e una chiave segreta (`CHIAVE_SEGRETA_MENSA_2026`). Questa stringa viene successivamente criptata tramite funzione di hashing per garantire la sicurezza del sistema e prevenire che malintenzionati possano riprodurre token validi tramite attacchi di tipo bruteforce. Una volta ricevuto il token dal server, la libreria QRCode.js viene utilizzata attraverso il costruttore `new QRCode` che, dato il token come parametro testuale, genera automaticamente il codice QR visuale con le dimensioni e il livello di

correzione degli errori specificati, permettendo così la visualizzazione del buono pasto in formato scannerizzabile.

Countdown di Scadenza Il token QR ha una validità di 5 minuti. Un countdown visualizza il tempo rimanente:

Listing 24: Funzione countdown

```
function avviaCountdown(scadenza) {
    const countdownEl = document.getElementById('countdown');
    const scadenzaDate = new Date(scadenza);

    const interval = setInterval(function () {
        const now = new Date();
        const diff = scadenzaDate - now;

        if (diff <= 0) {
            clearInterval(interval);
            countdownEl.textContent = 'SCADUTO';
            countdownEl.classList.remove('text-primary');
            countdownEl.classList.add('text-danger');
            return;
        }

        const minuti = Math.floor(diff / 60000);//conversione da millisecondi a
            minuti
        const secondi = Math.floor((diff % 60000) / 1000);//conversione da
            millisecondi a secondi

        countdownEl.textContent =
            String(minuti).padStart(2, '0') + ':' + //aggiunge zeri davanti se la
                stringa e' piu' corta di 2 caratteri
            String(secondi).padStart(2, '0');

        if (diff < 60000) {
            countdownEl.classList.remove('text-primary');
            countdownEl.classList.add('text-danger');
        }
    }, 1000);

    document.getElementById('qrModal')
        .addEventListener('hidden.bs.modal', function () {
        clearInterval(interval);
    });
}
```

La funzione calcola continuamente la differenza tra l'orario corrente e la scadenza del token, aggiornando il contatore ogni secondo per fornire un feedback in tempo reale all'utente. Quando il tempo residuo scende sotto il minuto, il colore del countdown cambia in rosso. Una volta che il tempo è terminato, il contatore mostra la scritta SCADUTO.

Polling per Verifica Utilizzo Token Un sistema di polling controlla ogni 2 secondi se il token è stato utilizzato, permettendo la chiusura automatica della modale:

Listing 25: Sistema di polling

```
function avviaPollingToken(modal) {
    if (pollingInterval) clearInterval(pollingInterval);

    //recupero numero di pasti iniziale
    fetch('procedure/controlla_token.php')
        .then(response => response.json())
        .then(data => {
            if (!data.success) return;
            const pastiIniziali = data.pasti;

            pollingInterval = setInterval(function () {
                fetch('procedure/controlla_token.php')
                    .then(response => response.json())
                    .then(data => {
                        if (data.success && data.pasti < pastiIniziali) {
                            clearInterval(pollingInterval);
                            pollingInterval = null;

                            // Aggiorna il numero di pasti nella card
                            document.querySelector(
                                '.display-3.fw-bold.text-primary'
                            ).textContent = data.pasti;

                            modal.hide();
                            mostraToast('Buono utilizzato',
                                'Il tuo buono pasto e stato validato!',
                                'success');
                        }
                    });
            }, 2000);
        });

    document.getElementById('qrModal')
        .addEventListener('hidden.bs.modal', function () {
```

```
        if (pollingInterval) {
            clearInterval(pollingInterval);
            pollingInterval = null;
        }
    }, { once: true });
}
```

Il sistema di polling inizia recuperando il numero iniziale di pasti come valore di riferimento per il confronto. Successivamente, attraverso chiamate periodiche ogni 2 secondi al server, verifica il numero attuale di pasti dello studente. Quando rileva una diminuzione del conteggio, interpreta questo cambiamento come una conferma che il token è stato validato con successo al tornello della mensa. A questo punto aggiorna automaticamente il contatore dei pasti visualizzato nella card dell'area personale, chiude la modale del QR code e mostra una notifica toast di successo per informare l'utente che il buono è stato utilizzato. Per garantire un uso efficiente delle risorse, il polling si interrompe automaticamente quando l'utente chiude manualmente la modale.

Sistema di Notifiche Toast Le notifiche vengono visualizzate tramite toast Bootstrap:

Listing 26: Funzione per mostrare toast

```
function mostraToast(titolo, messaggio, tipo = 'info') {
    document.getElementById('toastTitle').textContent = titolo;
    document.getElementById('toastMessage').textContent = messaggio;

    const toastEl = document.getElementById('alertToast');
    toastEl.classList.remove('bg-danger', 'bg-success',
                             'bg-warning', 'text-white');

    if (tipo === 'danger') {
        toastEl.classList.add('bg-danger', 'text-white');
    } else if (tipo === 'success') {
        toastEl.classList.add('bg-success', 'text-white');
    }

    const toast = new bootstrap.Toast(toastEl);
    toast.show();
}
```

Validazione token La pagina `tornello.php` è stata realizzata per simulare il funzionamento di un tornello fisico all'ingresso della mensa. La pagina è volutamente priva di sessione e di qualsiasi meccanismo di autenticazione: essendo concepita esclusivamente come simulazione, si è scelto di mantenerla aperta e accessibile direttamente tramite URL. È noto che in un sistema reale questo

componente richiederebbe un livello adeguato di protezione, ad esempio tramite autenticazione su rete locale o altri meccanismi di controllo degli accessi.

Al caricamento della pagina lo scanner QR si avvia automaticamente tramite la libreria `html5-qrcode`. Una volta inquadrato un QR code, lo scanner si blocca e il token decodificato viene inviato in formato JSON allo script `procedure/valida_token.php` tramite una richiesta `fetch` in `POST`. In base alla risposta del server, la pagina mostra un pannello di esito con un'animazione visiva distinta per accesso consentito o negato, indicando il nome dello studente e il tipo di pasto. Dopo cinque secondi il sistema si azzera automaticamente e lo scanner riparte, pronto per una nuova scansione.

Acquisto di nuovi pasti Lo studente, prima di procedere all'acquisto, è messo a conoscenza della sua fascia di ISEE e del costo per pasto che andrà a pagare. Il bottone "Carica pasti" porta al form di acquisto di questi ultimi contenuto nella pagina `pagamento.php`.

Form di acquisto La pagina `pagamento.php` contiene il form di acquisto fittizio dei pasti: presenta un menù a discesa per selezionare il circuito di pagamento, un input di tipo numerico contenente la quantità di pasti desiderata, un bottone per confermare il pagamento e una label che indica il prezzo che si andrà a pagare. Questa viene aggiornata in tempo reale grazie all'attributo onchange dell'input numerico. Come suggerisce il nome ogni volta che cambia il valore inserito, viene chiamata la funzione Javascript calcolaCosto alla quale viene passato il numero di pasti inserito e il costo per pasto associato allo studente in base alla sua fascia ISEE:

Listing 27: Funzione di calcolo del costo totale

```
function calcolaCosto(nPasti, costo) {
    document.getElementById("costo").innerText = "    " + (nPasti * costo).toFixed
        (2);
}
```

Al click del bottone "Procedi al pagamento", viene elaborata la richiesta dalla pagina `acquisto_pasto.php`.

Feedback post acquisto La pagina `acquisto_pasto.php` è incaricata di chiamare la procedura `inserisci_consumazione`: questa ha come parametri il codice fiscale dello studente, la fascia ISEE a cui appartiene e la quantità di pasti che vorrebbe acquistare. Se la procedura termina andando a buon fine viene dato un feedback positivo altrimenti negativo. In entrambi i casi si potrà tornare all'area personale tramite l'apposito bottone.

1.5 Area Operatore

1.5.1 Cartella includes

All'interno di tale cartella sono presenti alcuni file particolarmente immportanti per la funzionalità di questa sezione:

- `session_check.php` che svolge il controllo della sessione e del ruolo della persona che accede ad un pagina. Viene spesso richiamato con `require_once` all'interno di altri file
- `sidebar.php` gestisce la barra di navigazione laterale (Sidebar) per il pannello di controllo degli operatori/amministratori (vedi la sezione 1.5.4)

1.5.2 `frontend_login_operatori.php`

Per accedere all'area operatori è necessario navigare all'URL
`Sito/operatore/frontend_login_operatori.php`.
Tale pagina, se l'utente non si è ancora loggato, reindirizza al form di Bootstrap per il login.

Una volta raggiunta la pagina di login, viene effettuato un controllo sulla sessione attiva. Nel caso in cui il login sia già stato eseguito, l'utente verrà automaticamente reindirizzato, altrimenti viene visualizzato il form.

```php
<?php
session_start();

if (isset($_SESSION["logged_in"])) {
    if ($_SESSION["logged_in"]) {
        header("location: dashboard.php");
    } else {
        header("location: logout.php");
    }
}
?>
```

Viene poi utilizzata la classe `container` di Bootstrap per visualizzare al suo interno le informazioni necessarie per il login. Il login effettivo viene gestito con un bottone di tipo `submit` che invia i dati di accesso in POST a `accesso_operatori.php`. L'accesso funziona in un modo molto simile a quello progettato per gli studenti, le uniche differenze emerse sono l'implementazione di due variabili di sessione `logged_in` per verificare se l'utente è loggato e `ruolo` per distinguere tra operatore e admin.

1.5.3 accesso_operatori.php

Qui viene svolto il controllo delle credenziali per l'accesso all'area operatori.
Viene svolto un controllo delle credenziali di accesso tramite la funzione `controlloDB`. Se le credenziali sono vuote viene stampato a video un messaggio HTML. La funzione viene implementata tramite un JavaScript. `echo '<script>changeTitle();</script>';`

```
function controlloDB($nomeUtente, $password, $conn) {
                //controllo di sicurezza delle credenziali, controlla se
                    sono vuoti
                if(empty($nomeUtente) || empty($password)){
                    echo '<script>changeTitle();</script>';
                    echo "<h2 class=\"errore\">Errore nella compilazione
                        del form, compila il <a href=\"
                        frontend_login_operatori.php\">form di accesso</a>
                         e riprova.</h2>";
                exit();
            }
```

Per l'accesso viene utilizzata una stored procedure per garantire atomicità e ridurre il rischio di SQL injection.

```
$query = "CALL login_operatore(:nomeUtente)";
                try {
                    $stmt = $conn->prepare($query);
                    $stmt->bindParam(':nomeUtente', $nomeUtente, PDO::
                        PARAM_STR);
                    if ($stmt->execute()) {
                        $result = $stmt->fetch(PDO::FETCH_ASSOC);
                        if($result) {
                            if(password_verify($password, $result['
                                passwordhash'])) {
                                session_unset();
                                $_SESSION['agent'] = sha1($_SERVER['
                                    HTTP_USER_AGENT']);
                                $_SESSION['nomeUtente'] = $nomeUtente;
                                $_SESSION['ruolo'] = $result['ruolo'];
                                $_SESSION['logged_in'] = true;
                                header("Location: dashboard.php");
                                exit();}
```

1.5.4 sidebar.php

La caratteristica principale di questo file è l'implementazione di un design responsive per il pannello di controllo su dispositivi mobili.

```
<div class="d-md-none p-3 bg-light border-bottom">
    <button ... data-bs-toggle="offcanvas" ...>
        <i class="bi bi-list"></i> Menu
    </button>
</div>
```

La classe `d-md-none` nasconde questo blocco su schermi di medie e grandi dimensioni, rendendolo visibile esclusivamente su dispositivi mobili.

```
<div class="offcanvas offcanvas-start d-md-none" ... id="sidebarOffcanvas"
    >
```

Si tratta di un menu laterale che scorre verso l'interno dello schermo da sinistra quando viene attivato, e che contiene gli stessi collegamenti di navigazione presenti nella versione per PC.

La navigazione tra le diverse sezioni è gestita dalle seguenti righe di codice, che vengono replicate per ciascuna sezione disponibile.

```
<a href="studenti.php" class="<?php echo $current_page === 'studenti' ? '
    active' : ''; ?>">
            <i class="bi bi-people-fill me-2"></i>Studenti
        </a>
```

La voce di menu attiva viene determinata tramite l'assegnazione della classe `active` alla sezione corrente. Quando ci si trova sulla pagina `studenti.php`, la variabile `$current_page` assume il valore `studenti` e, di conseguenza, viene applicata la classe `active` al collegamento corrispondente, evidenziando l'HTML della sezione dedicata.
Il bottone per poter navigare tra le sezioni viene generato dall'anchor tag `<a>`. Questo è un link ipertestuale che viene poi stilizzato tramite CSS.

1.5.5 `studenti.php`

Le funzionalità che sono state implementate per questa sezione sono:

- Ricerca degli studenti tramite stored procedure `ricerca_studenti(:campo, :valore)`.
- Aggiunta di una certa quantità di consumazioni tramite procedura richiamata in `aggiungi_pasto.php`.
- Rimozione di un pasto tramite procedura richiamata in `rimuovi_pasto.php`.

Una volta entrati nella pagina viene visualizzato il form per la ricerca, la tabella viene visualizzata solo se si trova un risultato.

```
// Eseguo la query
if ($stmt->execute()) {
    // Recupero tutti i risultati
    $risultati = $stmt->fetchAll(PDO::FETCH_ASSOC);
    // Controllo se ci sono risultati
    if (count($risultati) == 0) {
        $messaggio = '<div class="alert alert-warning">Nessuno
            studente trovato con i criteri di ricerca
           specificati.</div>';
    } else {
        $messaggio = '<div class="alert alert-success">Trovati
            ' . count($risultati) . ' studente/i.</div>';
    }
}
```

Generazione tabella La generazione della tabella avviene tramite un controllo sulla variabile `$risultati`. Se questa risulta > 0 allora viene generata la tabella.

```
<?php if ($risultati && count($risultati) > 0): ?>
    <div id="studenti-table-container" class="table-responsive">
```

La tabella viene popolata dinamicamente tramite l'uso di un `foreach`. L'istruzione prende l'array `$risultati`, contenente i dati di tutti gli studenti estratti dal database, e genera le righe per ogni studente.
Le celle vengono riempite con i vari attributi degli studenti. Per poterle riempire in maniera sicura viene utilizzato `htmlspecialchars`. Questo comando effettua un controllo di sicurezza che converte i caratteri speciali in entità HTML sicure.

```
<tbody id="studenti-table-body">
    <?php foreach ($risultati as $row): ?>
        <tr>
            //inserimento dati nelle celle
            <td><?php echo htmlspecialchars($row['cf']); ?></td>
            ...
            //dati dello studente
            ...
            //bottone modale
            ...
            //gestione bottone modale
        </tr>
    <?php endforeach; ?>
</tbody>
```

Pulsante modale All'interno di questa tabella è stato inserito un pulsante funzionale all'apertura di una finestra modale. Alla pressione del tasto, viene visualizzato un popup per la gestione dei pasti relativi allo studente della riga selezionata.
L'apertura della finestra viene relizzata tramite la classe `data-bs-toggle="modal"`.
L'aspetto fondamentale risiede nell'attributo `data-bs-target`: la sua specifica sintassi consente di indirizzare la chiamata alla finestra modale specifica di ciascuno studente. Il collegamento avviene tramite un identificativo (ID) univoco, generato concatenando un prefisso fisso (`modalStudente`) e una parte variabile univoca, corrispondente al codice fiscale (`cf`).

```
<button type="button" class="btn btn-sm btn-info" data-bs-toggle="modal"
    data-bs-target="#modalStudente<?php echo $row['cf']; ?>">Modifica
</button>
```

Affinché il meccanismo funzioni correttamente, questo codice deve essere accompagnato dalla definizione della struttura della finestra modale dinamica corrispondente:

```
 <div class="modal fade" id="modalStudente<?php echo $row['cf']; ?>"
</button>
```

All'interno della finestra modale sono presenti due form:

- **Inserimento consumazione** Tramite POST , alla pressione del bottone, viene richiamata una stored procedure tramite `aggiungi_pasto.php`.
- **Rimozione consumazione** Tramite POST , alla pressione del bottone, viene richiamata una stored procedure tramite `rimuovi_pasto.php`.

Entrambi i form passano un `input type="hidden"` con valore `"<?php echo $row['cf']; ?>"`

1.5.6 `notizie.php`

Le funzionalità che sono state implementate per questa sezione sono:

- Visualizzazione delle notizie sul database.
- Attivazione e disattivazione delle notizie.
- Inserimento di notizie lato database.

All'interno di tale sezione vengono visualizzate due card in colonna.

```
<div class="col-lg-5 mb-4">
    <div class="card">
</button>
```

Nella prima è presente il form per l'inserimento delle notizie, nella seconda la visualizzazione delle notizie sul database.

Visualizzazione notizie All'apertura della sezione, il sistema esegue immediatamente la chiamata alla procedura per popolare l'interfaccia utente senza richiedere azioni aggiuntive.

Attivazione e disattivazione notizie L'entità notizia è caratterizzata dall'attributo `'attiva'`, introdotto per gestirne la visibilità pubblica. Attraverso lo script `toggle_notizia.php`, viene invocata una procedura che consente di abilitare o inibire la visualizzazione della notizia all'interno del sito.
Analogamente a quanto implementato nella sezione studenti, i pulsanti di azione vengono generati dinamicamente tramite un ciclo iterativo. A differenza del caso precedente, l'interazione non prevede l'apertura di una finestra modale, bensì l'invio di un campo di input nascosto (`hidden`) allo script di elaborazione. Tale campo contiene l'identificativo univoco (`id`) necessario per individuare la specifica notizia su cui operare.

```
<input type="hidden" name="id_notizia" value="<?php echo $row1['id']; ?>">
```

Inserimento di notizie Il form di inserimento della notizia presenta diversi campi. Per la redazione del contenuto è stata implementata una funzionalità di editor di testo avanzato (Rich Text Editor). L'interfaccia è stata realizzata integrando le componenti grafiche di Bootstrap con le funzionalità native di HTML5 e la logica JavaScript.

Tralasciando l'aspetto puramente grafico, l'elemento tecnico centrale è la seguente linea di codice:

```
<div id="editor-contenuto" contenteditable="true" class="form-control"
    ...></div>
<textarea id="contenuto-notizia" name="contenuto" style="display: none;"
    ></textarea>
```

L'attributo `contenteditable="true"` è una funzionalità nativa di HTML5 che rende un elemento `div` editabile dall'utente, permettendo di visualizzare in tempo reale la formattazione applicata.

All'interno della toolbar, ai vari pulsanti sono associate specifiche funzioni JavaScript:

```
onclick="formatText('bold')" title="Grassetto">
onclick="formatText('italic')" title="Corsivo">
onclick="formatText('underline')" title="Sottolineato">
onclick="formatText('insertUnorderedList')"
onclick="formatText('insertOrderedList')"
```

Alla pressione del pulsante, queste funzioni applicano il tag HTML corrispondente (es. `<b>` o `<strong>`) alla porzione di testo selezionata all'interno dell'area editabile. Tuttavia, poiché i dati contenuti in un elemento `div` non vengono trasmessi automaticamente durante il submit di un form HTML, è stato necessario implementare un meccanismo di sincronizzazione. È stata inserita una `textarea` nascosta (`display: none`) che funge da ponte tra l'editor visivo e il backend. Uno script JavaScript, in ascolto sull'evento di invio del form, copia il contenuto HTML generato nel `div` all'interno della `textarea` nascosta, garantendo così il corretto passaggio dei dati alla procedura PHP di salvataggio.

Importazione immagini Lato Frontend per importare le immagini viene utilizzato una tipologia di input diversa

```
<input type="file" class="form-control" id="immagine-notizia"
       name="immagine" accept="image/jpeg,image/png,image/jpg">
```

L'attributo più importante è l'etichetta `name="immagine"` in quanto verrà utilizzato lato php per trovare questo file all'interno dell'array globale `$_FILES`. Un'aspetto da approfondire è la gestione dei percorsi.

Un secondo attributo molto importante è `enctype="multipart/form-data"`. Solitamente quando

si inviano dati viene utilizzato un metodo di codifica dei dati di default. Questo metodo trasforma tutto in testo. Per poter inviare i file, che sono dati binari complessi, è necessario mantenere la forma binaria. Nel foom di inserimento delle notizie viene quindi aggiunto questa tipologia di encoding dei dati.

```
<form id="form-notizia" method="POST" action="..\procedure\
    aggiungi_notizia.php" style="display: inline;" enctype="multipart/form
    -data">
```

Osserviamo ora alcune particolarità della procedura `aggiungi_notizia.php`.
La prima variabile è il percorso fisico sul server relativo allo script PHP corrente. Si sale di una cartella con `../` e poi si accede alla cartella `notizie`. Serve a PHP per sapere dove salvare il file fisicamente.

```
        $percorsorisorenotizie = "../risorse/notizie/";
        $percorsorisorenotizieupload = "risorse/notizie/";
```

La seconda variabile è il percorso web che verrà salvato nel database.
Si controlla se la cartella di destinazione esiste effettivamente sul sito.

```
        if (!is_dir($percorsorisorenotizie)) {
            echo 'la directory non esiste';
        }
```

In seguito si controlla se i dati relativi all'immaginesono stati inviati con il `name="immagine"` nell'array `$_FILES`. Se il form è stato inviato si recuperano i dati relativi all'immagine e li assegna a delle variabili.
Tramite `$tmpName` si salva il percorso temporaneo dove PHP ha
Con `basename($name)` si rimuovono eventuali percorsi dal nome del file. In quetso modo si impedisce ad eventuali hacker di finire in cartelle di sistema sensibili.

```
        if (isset($_FILES['immagine']) && $_FILES['immagine']['error'] ==
            0) {
            $name = $_FILES['immagine']['name'];
            $tmpName = $_FILES['immagine']['tmp_name'];
            $nomefile = basename($name); //rende il nome sicuro
```

Il file viene salvato effettivamenete nel `$percorso`, nel database si riporta `$percorsoupload`.

```
            $percorso = $percorsorisorenotizie . $nomefile;
            $percorsoupload = $percorsorisorenotizieupload . $nomefile;
        }
```

`move_uploaded_file(da, a)` Prende il file dalla cartella temporanea ($tmpName) e lo sposta definitivamente nella cartella di destinazione ($percorso).

```
if (move_uploaded_file($tmpName, $percorso)) {
            echo "File caricato con successo in: " . $percorso;
        } else {
            echo "Errore durante il caricamento del file.";
        }
```

1.5.7 editor.js

Questo script costituisce il motore logico che rende operativo l'editor di testo precedentemente analizzato.

```
function formatText(command) {
    document.execCommand(command, false, null);
    document.getElementById('editor-contenuto').focus();
}
```

Formattazione del testo La funzione principale, `formatText`, gestisce la formattazione del contenuto. Alla pressione dei pulsanti della toolbar, viene invocato il metodo nativo `document.execCommand`, il quale applica automaticamente i tag HTML corrispondenti (es. grassetto, corsivo) alla porzione di testo selezionata dall'utente. Contestualmente, viene garantita l'usabilità attraverso il metodo `.focus()`: questa istruzione riporta immediatamente il cursore attivo all'interno dell'area di editazione, evitando che l'utente debba selezionarla nuovamente manualamente dopo ogni click.

Sincronizzazione e invio dati Poiché gli elementi `div` con attributo `contenteditable` non fanno parte degli elementi di input standard dei form HTML, il loro contenuto non viene incluso automaticamente nella richiesta HTTP POST al momento dell'invio.
Per risolvere questa limitazione tecnica, è stato implementato un listener sull'evento `submit` del form.

Il codice seguente intercetta il momento esatto in cui l'utente preme il pulsante di invio e svolge tre operazioni in sequenza immediata:

1. Recupera l'intero contenuto HTML (inclusi i tag di formattazione come `<b>` o `<ul>`) dal `div` dell'editor.

2. Trasferisce tale contenuto nel `value` della `textarea` nascosta (`contenuto-notizia`).

3. Lascia che il form proceda con l'invio standard, trasportando ora i dati corretti.

Da notare l'utilizzo dell'operatore di *optional chaining* (`?.`), che previene errori JavaScript nel caso in cui lo script venga caricato in pagine dove il form non è presente.

```
document.getElementById('form-notizia')?.addEventListener('submit',
    function (e) {
    const editorContent = document.getElementById('editor-contenuto').
        innerHTML;
    document.getElementById('contenuto-notizia').value = editorContent;
});
```

Validazione della lunghezza e vincoli del database Al fine di prevenire errori in fase di inserimento nel database (errore di *Data Truncation*), è stato implementato un controllo lato client che agisce in tempo reale mentre l'utente digita. Il sistema database MySQL/MariaDB utilizza per il campo contenuto il tipo di dato `TEXT`, il quale possiede un limite fisico di archiviazione pari a 65.535 caratteri.

Lo script seguente aggiunge un listener sull'evento `input`: ad ogni tasto premuto, viene calcolata la lunghezza del solo testo visibile (ignorando i tag HTML tramite `innerText`). Qualora tale lunghezza superi la soglia massima consentita, il sistema:

1. Avvisa l'utente tramite un alert.

2. Tronca automaticamente il contenuto eccedente per rientrare nei limiti consentiti.

```
document.getElementById('editor-contenuto')?.addEventListener('input',
    function () {
    const text = this.innerText || this.textContent;
    if (text.length > 65535) {
        alert('Il contenuto ha raggiunto il limite massimo di 65535
            caratteri');
        this.innerHTML = this.innerHTML.substring(0, 65535);
    }
});
```

1.5.8 Menù

La pagina menu.php permette agli operatori di gestire tutto quello che riguarda la creazione, assegnazione e visualizzazione dei menu della mensa, inizialmente grazie alle chiamate delle procedure: get_piatti() (per l'elenco completo dei piatti disponibili), get_elenco_menu() (per la lista dei menù esistenti) e get_menu_programmati()(per i menù già assegnati a date future) recupero i dati, dopo di che si recuperano i dati e vengono convertiti in array php

visualizzazione dei menu: I menù sono essenzialmente rappresentati come liste di piatti salvate nel database, in modo da costituire dei preset che facilitano il lavoro dell'operatore al momento dell'assegnazione dei menù giornalieri. La card di visualizzazione viene popolata compilando ogni riga con il nome del menù e due pulsanti che fanno riferimento all'id del menù: il primo apre un modale che, ad ogni apertura, recupera dal database la lista dei piatti associati e la mostra nel corpo della finestra, mentre il secondo permette di eliminare il menù.

Creazione dei menu: La creazione di un nuovo menù ha richiesto particolare attenzione dal punto di vista dell'interfaccia: in un contesto realistico il database può contenere un numero elevato di piatti, e costruire uno strumento di selezione che fosse allo stesso tempo completo, rapido da usare e intuitivo non è stato di banale realizzazione. Il form di creazione di un nuovo menù è composto da un campo testo per il nome e da un sistema interattivo per la selezione dei piatti. Questo secondo elemento non è un semplice `select` multiplo, ma una casella di ricerca con suggerimenti dinamici, realizzata tramite lo script `menu_search.js`.

Il funzionamento dello script parte da una scelta architetturale precisa: l'intero elenco dei piatti, recuperato dal database lato PHP, viene convertito in un array JavaScript tramite `json_encode` e incorporato direttamente nella pagina. In questo modo la ricerca avviene completamente lato client, senza ulteriori richieste al server.

Quando l'operatore inizia a digitare nella casella di ricerca, la funzione `initSearch()` intercetta l'evento di input, confronta il testo digitato con i nomi dei piatti nell'array e mostra fino a venti risultati pertinenti in un elenco a tendina. I piatti già selezionati vengono esclusi automaticamente dai suggerimenti grazie a una `Map` JavaScript che tiene traccia delle selezioni correnti.

Quando l'operatore clicca su un piatto suggerito, la funzione `addItem()` lo aggiunge visivamente come badge colorato e cliccabile all'interno del form. All'interno di ogni badge è inserito un campo nascosto (`input type hidden`) con il nome `piatti[]` e il valore dell'id del piatto, che sarà poi trasmesso al server alla sottomissione del form. Cliccando sul badge, il piatto viene rimosso sia visivamente che dalla `Map` delle selezioni.

Alla sottomissione del form, il file `aggiungi_menu.php` raccoglie gli id dei piatti selezionati dall'array `POST`, li converte in interi per sicurezza e li unisce in un'unica stringa separata da virgole tramite `implode`. Questa stringa CSV viene passata come parametro alla procedura `aggiungi_menu(:nome, :piatti_csv)`, che si occupa di inserire il menù nel database e di associargli i piatti corrispondenti.

Assegnazione dei menù alle date La parte inferiore sinistra della pagina ospita un form dedicato all'assegnazione di un menù esistente a un giorno specifico. L'operatore seleziona la data tramite un *date picker*, sceglie il tipo di pasto (pranzo o cena, codificati rispettivamente con i valori `0` e `1`) e seleziona il menù da un menu a tendina popolato con i dati già caricati in `$menu_list`. Il form invia i dati a `assegna_menu_giorno.php`. I menù già programmati per date odierne o future vengono mostrati in una tabella dedicata, con la possibilità di eliminarli tramite `elimina_menu_giorno.php`, passando data e tipo pasto come campi nascosti.

1.5.9 Gestione Piatti

Le principali funzionalità implementate in questo modulo sono:

- Inserimento nuovi piatti: operazione gestita tramite la procedura invocata dallo script `aggiungi_piatto.php`.
- Ricerca piatti: effettuata mediante l'interrogazione al database tramite la procedura `ricerca_piatti(:id, :nome)`.

Dal punto di vista implementativo, entrambe le funzioni sono state sviluppate mantenendo la coerenza architetturale con il resto del progetto. Nello specifico, la logica di inserimento è analoga a quella della sezione *Notizie*, mentre il meccanismo di ricerca e filtraggio replica il pattern utilizzato per la gestione *Studenti*.

1.5.10 Gestione operatori

La sezione gestione operatori può essere visualizzata esclusivamente dagli admin.
Nella sidebar viene fatto il seguente controllo.

```
<?php if ($user_role === 'admin'): ?>
    <a href="operatori.php" class="<?php echo $current_page === 'operatori
        ' ? 'active' : ''; ?>">
        <i class="bi bi-shield-lock me-2"></i>Operatori
    </a>
<?php endif; ?>
```

Una volta che porte il operatori.php viene svolto un ulteriore controllo

```php
if ($user_role !== 'admin') {
    header('Location: dashboard.php');
    exit();
}
```

Le funzionalità che sono state implementate sono:

- Inserimento nuovi operatori tramite procedura richiamata in `aggiungi_operatore.php`
- Visualizzazione operatori tramite la procedura `get_operatori()`.
- Rimozione operatori tramite procedura richiamata in `elimina_operatore.php`

Anche in questo contesto, lo sviluppo ha privilegiato il riuso del codice e la coerenza dell'interfaccia. Dal punto di vista implementativo, sia la logica di inserimento dati che la struttura di visualizzazione tabellare ricalcano fedelmente i pattern architetturali già adottati nella sezione *Notizie*.

2 Database

2.1 Obiettivi del progetto

Gli obiettivi principali del progetto sono:

- Gestire tre tipologie di utenti (amministratore, operatore e studente) con memorizzazione sicura delle credenziali di accesso
- Mantenere un profilo completo per ogni studente con dati anagrafici, fiscali e numero di pasti disponibili
- Tracciare le transazioni di acquisto e utilizzo dei pasti per garantire trasparenza e controllo
- Pubblicare e gestire notizie con contenuti multimediali accessibili dalla home page
- Fornire un sistema differenziato di permessi per amministratori e operatori nella gestione del sistema

2.2 Analisi dei requisiti

Il sistema informatico modella la gestione di una mensa universitaria. Il sistema è accessibile tramite un sito web suddiviso in tre aree: un'area pubblica, un'area studente e un'area operatore.

Utenti del sistema. Il sistema prevede tre tipologie di utente: *studente*, *operatore* e *amministratore*. Ogni utente è identificato da un nome utente univoco e da una password. Il ruolo determina le operazioni che l'utente è autorizzato a compiere. L'amministratore dispone di tutti i privilegi dell'operatore e in aggiunta può gestire gli operatori del sistema.

Studente. Lo studente può registrarsi al sistema fornendo i propri dati anagrafici, il codice fiscale e la fascia ISEE di appartenenza. La fascia ISEE determina il costo unitario di ogni pasto. Lo studente può acquistare pasti, visualizzare il proprio saldo pasti e generare un token (codice QR) per accedere fisicamente alla mensa. Il token è temporaneo e associato a un singolo pasto non ancora utilizzato. L'utilizzo del token decrementa il saldo pasti dello studente.

Pasto e accesso alla mensa. Un pasto è un ticket digitale acquistato dallo studente. Ogni pasto registra la data di acquisto e, se utilizzato, la data di utilizzo. L'accesso fisico alla mensa avviene tramite la scansione del codice QR generato dallo studente. La validazione del codice QR marca il pasto come utilizzato e decrementa il saldo pasti.

Menu e piatti. Un menu è un insieme di piatti. I piatti sono le singole portate disponibili nella mensa; ogni piatto può essere associato a uno o più allergeni. I menu vengono creati dagli operatori

combinando i piatti disponibili. Ogni giorno possono essere assegnati al massimo due menu: uno per il pranzo e uno per la cena. L'insieme dei menu assegnati a una data specifica costituisce il menu del giorno.

Operatore e amministratore. L'operatore può gestire gli studenti, i pasti, i piatti, i menu e le notizie. La gestione degli studenti comprende la ricerca per codice fiscale, nome o email, l'aggiunta e la rimozione manuale di pasti. La gestione dei menu comprende la creazione di nuovi menu, l'assegnazione dei menu del giorno e la loro eliminazione. L'amministratore, in aggiunta, può aggiungere e rimuovere operatori dal sistema.

Notizie. Le notizie sono comunicazioni pubblicate sulla home page del sito. Ogni notizia è composta da un titolo, una descrizione, un contenuto testuale, un'immagine di copertina e una data di pubblicazione. Le notizie possono essere attivate o disattivate per controllarne la visibilità. La pubblicazione e la gestione delle notizie sono riservate agli operatori e all'amministratore.

Area pubblica. L'area pubblica è accessibile senza autenticazione. Permette la visualizzazione delle notizie attive e del menu del giorno, selezionabile per data.

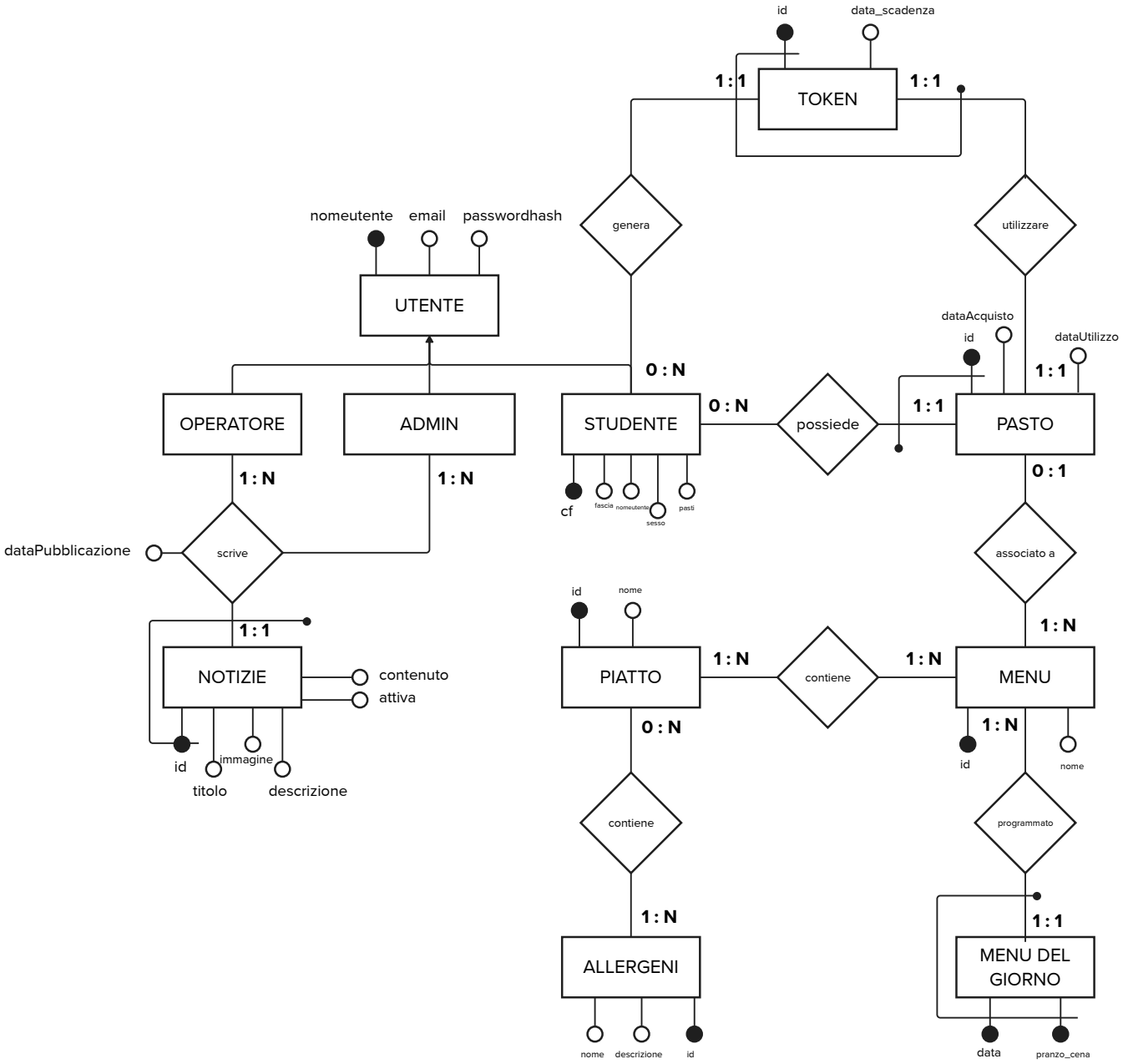
Termine	Definizione	Sinonimi	Relazioni
Studente	Utente che può acquistare e utilizzare i pasti, visualizzare i menu e le notizie	-	Pasto, Fascia
Operatore	Utente con privilegi di gestione che può visualizzare gli studenti, modificare i pasti, gestire menu e notizie	Addetto	Pasto, Menu, Piatto, Notizia, Studente
Amministratore	Utente con i massimi privilegi, può gestire operatori oltre a tutte le funzioni dell'operatore	Admin	Operatore, Studente, Pasto, Menu, Notizia
Pasto	Ticket digitale che consente l'accesso alla mensa per consumare il menu del giorno	Consumazione	Studente, Codice QR, Fascia, Menu del giorno
Menu	Insieme di piatti disponibili, utilizzato come modello per creare i menu del giorno	-	Piatto, Menu del giorno, Operatore
Menu del giorno	Menu specifico assegnato a una data e un pasto (pranzo o cena)	-	Menu, Pasto, Piatto
Piatto	Singola portata che compone un menu	Portata	Menu, Allergene
Notizia	Contenuto informativo pubblicato sulla home page del sito	Comunicazione	Operatore, Amministratore
Allergene	Sostanza che può causare reazioni allergiche, associata ai piatti	-	Piatto
Fascia	Categoria ISEE di appartenenza dello studente che determina il costo del pasto	Categoria ISEE	Studente, Pasto
Token	Identificatore univoco temporaneo associato ad un codice QR per la validazione	QR Code	Codice QR, Pasto, Studente

2.2.1 Glossario dei termini

2.2.2 Lista operazioni

Operazione	Descrizione
Operazione 1	Registrazione di un nuovo studente con tutti i dati anagrafici e fiscali (con media di 5 volte al giorno)
Operazione 2	Login di uno studente al sistema per accedere all'area personale (con media di 200 volte al giorno)
Operazione 3	Acquisto di pasti da parte dello studente specificando la quantità (con media di 100 volte al giorno)
Operazione 4	Generazione di un codice QR temporaneo per l'accesso alla mensa (con media di 150 volte al giorno)
Operazione 5	Validazione del codice QR e registrazione dell'utilizzo del pasto (con media di 150 volte al giorno)
Operazione 6	Visualizzazione del menu del giorno da parte degli studenti (con media di 300 volte al giorno)
Operazione 7	Ricerca di studenti da parte degli operatori per codice fiscale, nome o email (con media di 10 volte al giorno)
Operazione 8	Inserimento manuale di pasti da parte degli operatori per uno studente specifico (con media di 10 volte al giorno)
Operazione 9	Rimozione manuale di un pasto da parte degli operatori (con media di 10 volte al giorno)
Operazione 10	Creazione di un nuovo menu da parte degli operatori (con media di 5 volte al mese)
Operazione 11	Assegnazione di un menu a una data specifica e tipo pasto (pranzo/cena) (con media di 15 volte alla settimana)
Operazione 12	Aggiunta di un nuovo piatto con descrizione e allergeni (con media di 10 volte al mese)
Operazione 13	Ricerca di piatti per nome o ID (con media di 30 volte al mese)
Operazione 14	Pubblicazione di una nuova notizia con titolo, descrizione e contenuto (con media di 1 volta alla settimana)
Operazione 15	Attivazione o disattivazione di una notizia pubblicata (con media di 1 volta alla settimana)
Operazione 16	Eliminazione di una notizia dal sistema (con media di 1 volta al mese)
Operazione 17	Visualizzazione dell'elenco degli operatori da parte dell'amministratore (con media di 5 volte al mese)
Operazione 18	Aggiunta di un nuovo operatore da parte dell'amministratore (con media di 5 volte all'anno)
Operazione 19	Rimozione di un operatore da parte dell'amministratore (con media di 5 volte all'anno)
Operazione 20	Visualizzazione delle notizie attive nella home page (con media di 500 volte al giorno)
Operazione 21	Login di un'operatore o admin al pannello operatori (con media di 20 volte al giorno)

2.3 Progettazione concettuale: entità e attributi



2.3.1 Dizionario dei dati - Entità

Entità	Descrizione	Attributi	Identificatore
Utente	Entità padre che raccoglie i dati comuni di accesso per tutti gli attori del sistema.	nomeutente, email, password-hash	nomeutente
Studente	Specializzazione di Utente. Rappresenta il fruitore del servizio mensa.	cf, nome, cognome, sesso, data_nascita, indirizzo, citta, pasti	cf
Operatore	Specializzazione di Utente. Addetto alla gestione dei menu e delle notizie.	(Eredita gli attributi di Utente)	(Eredita nomeutente)
Admin	Specializzazione di Utente. Supervisore con privilegi completi.	(Eredita gli attributi di Utente)	(Eredita nomeutente)
Pasto	Rappresenta il diritto di consumazione acquisito da uno studente.	id, dataAcquisto, dataUtilizzo	id
Token	Codice univoco temporaneo generato per validare un pasto al tornello.	id, data_scadenza	id
Menu	Tipologia di menu disponibile (es. "Menu Invernale", "Menu Estivo").	id, nome	id
MenuDelGiorno	Pianificazione specifica che associa un Menu a una data e a un turno.	data, pranzo_cena	data, pranzo_cena
Piatto	Singola pietanza che può essere contenuta in un menu.	id, nome	id
Allergene	Sostanza allergenica che può essere contenuta in un piatto.	nome, descrizione	nome
Notizia	Comunicazione informativa scritta dallo staff (Operatore o Admin).	id, titolo, descrizione, contenuto, immagine, dataPubblicazione	id

2.3.2 Dizionario dei dati - Relazioni

Relazione	**Descrizione**	**Entità Coinvolte**
Assegnazione Ruolo	Associa un livello di permesso ad ogni account utente.	Utente, Ruolo
Profilo Studente	Collega l'account di accesso ai dati anagrafici dello studente (relazione 1:1).	Utente, Info_Studente
Appartenenza Fascia	Determina il costo del pasto in base alla fascia ISEE dello studente.	Info_Studente, Tariffa
Acquisto Pasto	Registra l'acquisto di un ticket da parte di uno studente.	Info_Studente, Pasto
Generazione Token	Associa un codice QR temporaneo a uno studente per l'accesso.	Info_Studente, Token
Validazione Token	Collega il token generato allo specifico pasto che verrà scalato.	Token, Pasto
Composizione Menu	Definisce quali piatti compongono un determinato menu (molti-a-molti).	Menu, Piatto
Presenza Allergeni	Indica quali allergeni sono contenuti in un piatto (molti-a-molti).	Piatto, Allergene
Pianificazione	Stabilisce quale menu viene servito in una data specifica (Menu del Giorno).	Menu, MenuDelGiorno
Pubblicazione	Associa una notizia all'autore (admin o operatore) che l'ha creata.	Notizia, Utente

Tabella 3: Dizionario dei dati: Relazioni.

2.3.3 Vincoli non esprimibili

ID	Descrizione del Vincolo
RV1	La rimozione manuale di una consumazione da parte dell'operatore è permessa solo se il numero di pasti è maggiore di zero.
RV2	L'utilizzo di una consumazione da parte dello studente è permesso solo se il numero di pasti residui è maggiore di zero.
RV3	L'aggiunta di nuovi operatori al sistema è riservata esclusivamente al ruolo *Admin*.
RV4	La rimozione di operatori esistenti è riservata esclusivamente al ruolo *Admin*.
RV5	L'inserimento di nuove notizie è permesso solo ai ruoli *Admin* e *Operatore*.
RV6	La disabilitazione (nascondere dalla home) di notizie è permessa solo ai ruoli *Admin* e *Operatore*.
RV7	La rimozione definitiva di notizie è permessa solo ai ruoli *Admin* e *Operatore*.
RV8	L'aggiunta di nuovi piatti all'archivio è permessa solo ai ruoli *Admin* e *Operatore*.
RV9	La creazione di nuovi menu (combinazioni di piatti) è permessa solo ai ruoli *Admin* e *Operatore*.
RV10	La visualizzazione dei dati sensibili degli studenti è permessa solo ai ruoli *Admin* e *Operatore*.
RV11	La pianificazione del calendario (menu del giorno) è riservata ai ruoli *Admin* e *Operatore*.
RV12	La generazione di un Token (QR Code) è una funzionalità riservata esclusivamente al ruolo *Studente*.
RV13	L'accesso fisico alla mensa tramite scansione token è possibile solo se il token è valido e non scaduto.

Tabella 4: Tabella delle regole di vincolo (Business Rules).

Le procedure utilizzate per esprimere tali vincoli sono approfondite nella sezione dedicata (vedi Sezione 2.7).

2.3.4 Regole di derivazione

ID	Descrizione della Regola
RD1	Quando viene inserito un nuovo record nella tabella *pasto* (acquisto di un ticket), il contatore dei pasti disponibili dello studente viene incrementato automaticamente.
RD2	Quando un pasto viene marcato come utilizzato (campo *dataUtilizzo* valorizzato), il contatore dei pasti disponibili dello studente viene decrementato automaticamente.
RD3	Quando un pasto non ancora utilizzato viene cancellato dal sistema, il contatore dei pasti disponibili dello studente viene decrementato automaticamente (annullando l'incremento iniziale).

Tabella 5: Tabella delle regole di derivazione

2.4 Ristrutturazione da concettuale a logico

Nel passaggio da schema concettuale a schema logico c'è stata la necessità di ristrutturare alcune entità.
Andando in ordine e seguendo le immagini qui sotto riportate possiamo contare cinque principali ristrutturazioni.
Nella figura qui di seguito la generalizzazione è stata gestita con un collasso verso l'alto ed un partizionamento. Le entità figlie `OPERATORE`, `ADMIN` e `STUDENTE` sono state eliminate. È stato aggiunta l'entità ruolo per distinguere i tre tipi. Le informazioni dello studente sono state associate ad una nuova entità `INFO_STUDENTE`.

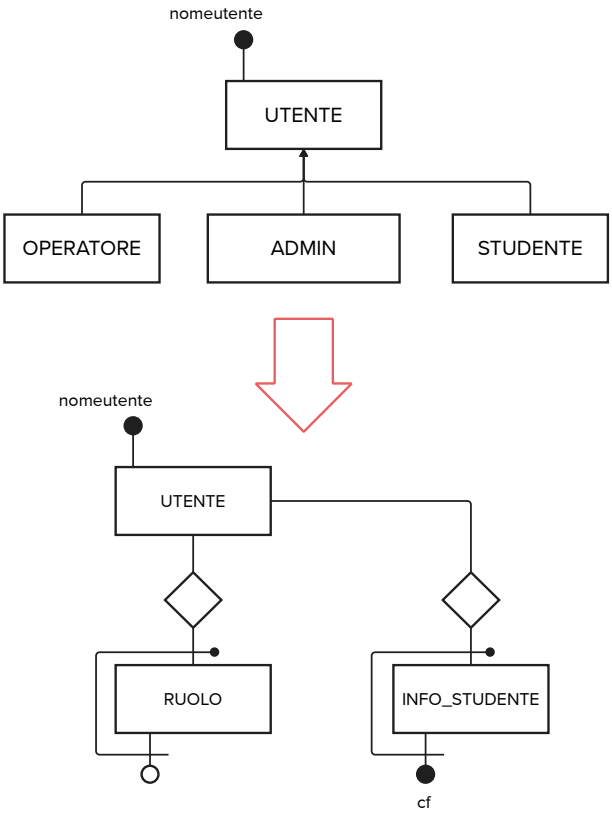


Figura 2: Accorpamento e risoluzione generalizzazione

In seguito alla precedente generalizzazione le relazioni tra `OPERATORE`, `ADMIN` e `NOTIZIA` è stata accorpata.

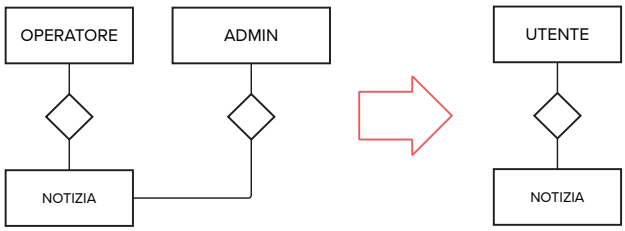


Figura 3: Accorpamento Relazioni

Le due relazioni successive sono molti a molti (N:N).Sono state risolte introducendo delle tabelle ponte.

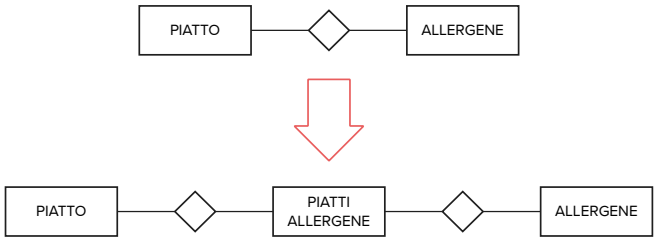


Figura 4: Risoluzione molti a molti

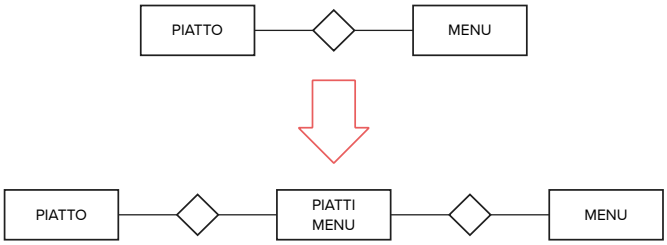


Figura 5: Risoluzione molti a molti

In questa figura si mostra l'esternalizzazione dell'attributo `fascia`. Questa è una normalizzazione o passaggio alla forma 3NF.

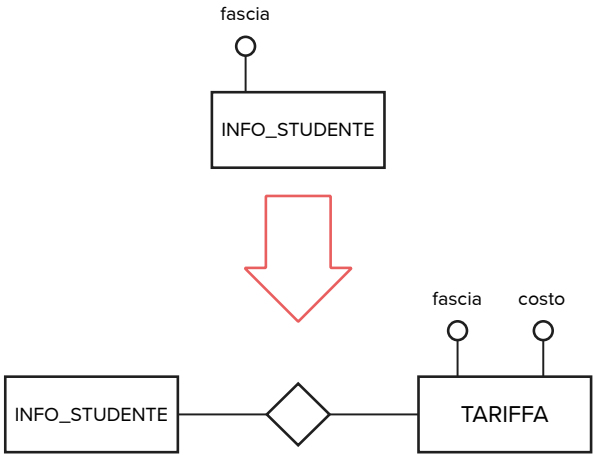


Figura 6: Normalizzazione

2.5 Progettazione logica

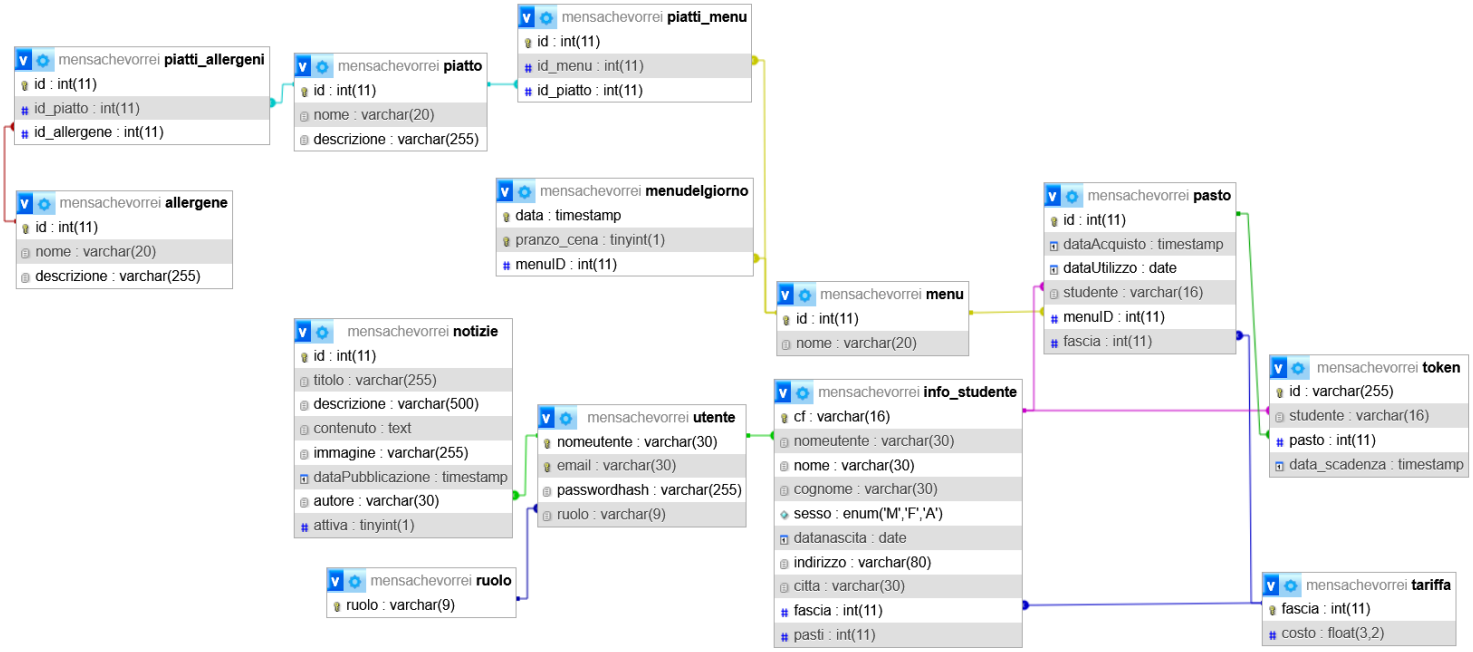


Figura 7: Enter Caption

- **ruolo**
  - `ruolo`: PK, varchar(9); Identificativo univoco del ruolo (es. admin, operatore, studente);
- **utente**
  - `nomeutente`: PK, varchar(30); Codice univoco per l'accesso al sistema;
  - `email`: varchar(30); Email univoca dell'utente;
  - `passwordhash`: varchar(255); Password cifrata per l'autenticazione;
  - `ruolo`: FK, varchar(9); Riferimento al ruolo assegnato all'utente;
- **tariffa**
  - `fascia`: PK, int(11); Identificativo numerico della fascia di reddito (ISEE);
  - `costo`: float(3,2); Costo del singolo pasto associato alla fascia;
- **info_studente**

- `cf`: PK, varchar(16); Codice identificativo dello studente;
- `nomeutente`: FK, varchar(30); Riferimento all'account utente associato;
- `nome`: varchar(30); Nome dello studente;
- `cognome`: varchar(30); Cognome dello studente;
- `sesso`: enum('M','F','A'); Genere dello studente;
- `datanascita`: date; Data di nascita dello studente;
- `indirizzo`: varchar(80); Indirizzo di residenza;
- `città`: varchar(30); Città di residenza;
- `fascia`: FK, int(11); Riferimento alla fascia tariffaria di appartenenza;
- `pasti`: int(11); Saldo dei pasti acquistati e non ancora consumati;

- **pasto**
  - `id`: PK, int(11); Identificativo univoco del ticket pasto;
  - `dataAcquisto`: timestamp; Data e ora in cui è stato acquistato il pasto;
  - `dataUtilizzo`: date; Data in cui il pasto è stato consumato (NULL se non utilizzato);
  - `studente`: FK, varchar(16); Studente che ha effettuato l'acquisto;
  - `menuID`: FK, int(11); Riferimento al tipo di menu scelto;
  - `fascia`: FK, int(11); Fascia tariffaria applicata al momento dell'acquisto;

- **Token**
  - `id`: PK, varchar(255); Codice univoco (es. QR Code) per la validazione;
  - `studente`: FK, varchar(16); Studente che ha generato il Token;
  - `pasto`: FK, int(11); Riferimento allo specifico pasto da scalare;
  - `data_scadenza`: timestamp; Momento di scadenza della validità del token;

- **notizie**
  - `id`: PK, int(11); Identificativo univoco della notizia;
  - `titolo`: varchar(255); Titolo della notizia;
  - `descrizione`: varchar(500); Breve descrizione o sottotitolo;
  - `contenuto`: text; Testo completo della notizia (può contenere HTML);
  - `immagine`: varchar(255); Percorso dell'immagine di copertina;

- `dataPubblicazione`: timestamp; Data e ora di pubblicazione;
- `autore`: FK, varchar(30); Utente che ha pubblicato la notizia;
- `attiva`: tinyint(1); Flag di visibilità (1=attiva, 0=nascosta);

- **menu**
  - `id`: PK, int(11); Identificativo univoco del menu;
  - `nome`: varchar(20); Nome identificativo del menu (es. "Menu Invernale");

- **menudelgiorno**
  - `data`: PK, timestamp; Data in cui viene servito il menu;
  - `pranzo_cena`: PK, tinyint(1); Indicatore del servizio (0 per pranzo, 1 per cena);
  - `menuID`: FK, int(11); Riferimento al menu pianificato per quella data;

- **piatto**
  - `id`: PK, int(11); Identificativo univoco del piatto;
  - `nome`: varchar(20); Nome della pietanza;
  - `descrizione`: varchar(255); Descrizione degli ingredienti o preparazione;

- **piatti_menu**
  - `id`: PK, int(11); Identificativo univoco della relazione;
  - `id_menu`: FK, int(11); Riferimento al menu contenitore;
  - `id_piatto`: FK, int(11); Riferimento al piatto incluso nel menu;

- **allergene**
  - `id`: PK, int(11); Identificativo univoco dell'allergene;
  - `nome`: varchar(20); Nome dell'allergene (es. Glutine, Lattosio);
  - `descrizione`: varchar(255); Note aggiuntive sull'allergene;

- **piatti_allergeni**
  - `id`: PK, int(11); Identificativo univoco della relazione;
  - `id_piatto`: FK, int(11); Riferimento al piatto;
  - `id_allergene`: FK, int(11); Riferimento all'allergene presente nel piatto;

2.5.1 Tavola delle operazioni

Op	Descrizione	Freq.	Tipo	Tabelle coinvolte
1	Registrazione nuovo studente	5/gg	INSERT	utente, info_studente
2	Login studente	200/gg	SELECT	utente, info_studente
3	Acquisto pasti	100/gg	INSERT	pasto, info_studente
4	Generazione QR/Token	150/gg	INSERT	token, pasto, info_studente
5	Validazione e consumo pasto	150/gg	UPDATE	pasto, info_studente
6	Vis. menu del giorno	300/gg	SELECT	menudelgiorno, menu, piatto
7	Ricerca studenti	50/gg	SELECT	info_studente
8	Inserimento pasti manuale	20/gg	INSERT	pasto, info_studente
9	Rimozione pasto manuale	10/gg	UPDATE	pasto, info_studente
10	Creazione menu	3/gg	INSERT	menu, piatti_menu
11	Pianificazione menu	7/gg	INSERT	menudelgiorno
12	Aggiunta piatto	2/gg	INSERT	piatto
13	Ricerca piatti	30/gg	SELECT	piatto
14	Pubblicazione notizia	1/sett	INSERT	notizie
15	Attivazione/Disatt. notizia	1/sett	UPDATE	notizie
16	Eliminazione notizia	1/gg	DELETE	notizie
17	Vis. lista operatori	5/anno	SELECT	utente, ruolo
18	Aggiunta operatore	2/anno	INSERT	utente
19	Rimozione operatore	2/anno	DELETE	utente
20	Vis. notizie home	500/gg	SELECT	notizie
21	Login operatore/admin	20/gg	SELECT	utente

2.5.2 Gestione ridondanze e relative tavole degli accessi

Nel database è presente una ridondanza controllata: infatti l'attributo *pasti* della tabella *info_studente* memorizza il numero di pasti disponibili per ogni studente. Questo valore potrebbe essere calcolato dinamicamente contando i record dove *dataUtilizzo* `IS NULL`. Invece si è scelto di mantenere la ridonza per poter ridurre gli accessi al database. Di seguito sono riportate due tabelle: la prima riporta le operazioni per l'inserimento di un pasto con ridondanza, la seconda senza.
(è possibile consultare i trigger nella sezione 2.9)

Concetto	**Costrutto**	**Accessi**	**Tipo**
pasto	E	1	INSERT
info_studente	E	1	UPDATE

Tabella 6: Inserimento pasti con ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
pasto	E	1	INSERT

Tabella 7: Inserimento pasti senza ridondanza

Per poter comprendere gli effetti di questa ridondanza è possibile visualizzare nelle prossime tabelle il numero di accessi che viene risparmiato in fase di visualizzazione del numero dei pasti.

Concetto	**Costrutto**	**Accessi**	**Tipo**
info_studente	E	1	SELECT

Tabella 8: Visualizzazione pasti disponibili con ridondanza

Concetto	**Costrutto**	**Accessi**	**Tipo**
pasto	E	N	SELECT
info_studente	E	1	SELECT

Tabella 9: Visualizzazione pasti disponibili con ridondanza

2.5.3 Conclusione analisi degli accessi

L'introduzione dell'attributo ridondante *pasti* nella tabella *info_studente* nasce dall'analisi dei volumi di accesso al database. Il sistema deve gestire un alto numero di letture del saldo pasti (Login e Generazione Token) rispetto alle operazioni di scrittura (Acquisto).

Definiamo N come il numero medio di pasti attivi (non consumati) presenti nello storico di uno studente. Ipotizzando un valore medio di $N = 15$ (dimensione media di un carnet pasti), confrontiamo i due scenari:

- **Senza ridondanza:** Per conoscere il saldo è necessario eseguire una `COUNT(*)` sulla tabella *pasto*. Questo comporta l'accesso a N record per ogni lettura.
- **Con ridondanza:** Il saldo è un semplice attributo intero nella tabella *info_studente*, richiedendo 1 solo accesso in lettura. In contropartita, ogni acquisto richiede 2 accessi (INSERT + UPDATE).

Di seguito è riportata la stima degli accessi giornalieri basata sulle frequenze operative previste:

Oper.	**Freq. GG**	**Acc. (Con ridondanza)**	**Acc. (Senza ridondanza)**
Op. 3 (Acquisto Pasti)	100	$100 \times 2 = 200$ (W)	$100 \times 1 = 100$ (W)
Op. 2 (Login Studente)	200	$200 \times 1 = 200$ (R)	$200 \times 15 = 3000$ (R)
Op. 4 (Generazione QR)	150	$150 \times 1 = 150$ (R)	$150 \times 15 = 2250$ (R)
TOT. GG		**550 accessi**	**5350 accessi**

Tabella 10: Confronto carico di lavoro sul Database ($N = 15$)

2.5.4 Tavola dei volumi

Tabella	**Volume stimato**	**Nota**
utente	2.600 record	Totale utenti registrati nel sistema
utente: Studente	2.500 record	Utenti che acquistano e consumano pasti (Crescita: ∼1.250/anno)
utente: Operatore	10 record	Personale addetto alla gestione menu e studenti
utente: Admin	5 record	Supervisori tecnici e gestione operatori
info_studente	2.500 record	Dati anagrafici e fiscali
pasto	37.500 record	Storico consumazioni e acquisti (Stima: ∼150 op./giorno)
token	200 record	Codici QR attivi (Volatile: vengono cancellati dopo l'uso)
piatto	200 record	Archivio delle pietanze disponibili in cucina
menu	50 record	Combinazioni predefinite di piatti
piatti_menu	200 record	Relazione tra menu e piatti contenuti
menudelgiorno	600 record	Calendario pianificato (2 pasti/gg × giorni lavorativi)
notizie	30 record	Comunicazioni pubblicate in Home Page
allergene	14 record	Lista statica allergeni (Normativa EU)
piatti_allergeni	300 record	Associazione piatti-allergeni

Tabella 11: Stima dei volumi dei dati su base annua.

2.6 Valutazione e normalizzazione

2.6.1 Normalizzazione

Tutte le tabelle del database rispettano la **Terza Forma Normale (3NF)** e, come si mostrerà, anche la **Forma Normale di Boyce-Codd (BCNF)**. Di seguito si verifica il soddisfacimento delle condizioni richieste per ciascuna forma normale.

Prima Forma Normale (1NF) Ogni tabella possiede una chiave primaria, tutti gli attributi contengono valori atomici e non sono presenti gruppi ripetuti. La relazione molti-a-molti tra `menu` e `piatto`, ad esempio, è stata risolta con la tabella ponte `piatti_menu`, evitando qualsiasi forma di attributo multivalore.

Seconda Forma Normale (2NF) La 2NF richiede che ogni attributo non chiave dipenda *funzionalmente* dall'intera chiave primaria. Nelle tabelle con chiave primaria semplice (come `utente`, `info_studente`, `pasto`, `token`, `menu`, `piatto`, `notizie`, `piatti_menu`, `piatti_allergeni`) questa condizione è soddisfatta per costruzione, in quanto tutte dispongono di una chiave `id` e tutti gli attributi dipendono direttamente da essa. L'unica tabella con chiave composta è `menudelgiorno` (chiave: `data`, `pranzo_cena`): presenta un unico attributo non chiave `"menuID"`, che dipende dall'intera coppia di valori e non da una sola parte di essa. La 2NF è quindi rispettata in tutte le tabelle.

Terza Forma Normale (3NF) La 3NF richiede l'assenza di dipendenze transitive, ovvero nessun attributo non chiave deve dipendere da un altro attributo non chiave. Il caso più rilevante riguarda l'attributo `fascia` in `info_studente`: il costo del pasto dipende dalla fascia ISEE, non direttamente dallo studente. Per eliminare questa dipendenza transitiva, il costo è stato esternalizzato nella tabella `tariffa`, che associa ogni fascia al relativo importo. Questa ristrutturazione è già stata illustrata nella sezione precedente. Nelle restanti tabelle non sono presenti dipendenze transitive: ogni attributo non chiave dipende direttamente e unicamente dalla chiave primaria della propria tabella.

Forma Normale di Boyce-Codd (BCNF) La BCNF richiede che ogni determinante di una dipendenza funzionale sia una superchiave. Essa si distingue dalla 3NF solo in presenza di più chiavi candidate sovrapposte, caso che non si verifica in nessuna tabella di questo schema. In tutte le tabelle l'unico determinante di ogni dipendenza funzionale è la chiave primaria, che è per definizione una superchiave. Ne consegue che tutte le tabelle soddisfano anche la BCNF.

l'unica eccezione riguarda l'attributo `pasti` in `info_studente`, che rappresenta una ridondanza controllata e consapevole, discussa in dettaglio nella sezione dedicata (vedi Sezione 2.5.2).

2.7 Procedure di gestione generale

2.7.1 get_menu_programmati

La procedura permette di visualizzare i menù programmati per un determinato giorno e per uno dei due servizi disponibili (*pranzo* o *cena*). Come parametro viene passata una data specifica; la procedura restituisce il nome del menù e i dettagli di ogni piatto associato.

Viene utilizzata la funzione `GROUP_CONCAT` insieme al `GROUP BY` poiché un piatto può avere più allergeni: questi vengono aggregati in un'unica stringa separata da virgole. Tramite i `JOIN` si recuperano le informazioni necessarie del menù per la data richiesta. Il `GROUP BY` raggruppa per una serie di attributi in modo da ottenere, per ogni singolo piatto, una sola riga contenente tutti i suoi allergeni.

```
1  CREATE PROCEDURE `get_menu_programmati` ()
2  BEGIN
3      SELECT mdg.data, CASE WHEN mdg.pranzo_cena = 0 THEN 'Pranzo' ELSE 'Cena' END AS tipo_pasto, m.nome AS
       ↪  menu_nome, p.id AS piatto_id, p.nome AS piatto_nome, p.descrizione AS piatto_descrizione,
       ↪  GROUP_CONCAT(a.nome SEPARATOR ', ') AS allergeni
4      FROM menudelgiorno mdg
5      INNER JOIN menu m ON mdg.menuID = m.id
6      INNER JOIN piatti_menu pm ON m.id = pm.id_menu
7      INNER JOIN piatto p ON pm.id_piatto = p.id
8      LEFT JOIN piatti_allergeni pa ON p.id = pa.id_piatto
9      LEFT JOIN allergene a ON pa.id_allergene = a.id
10     WHERE DATE(mdg.data) = dataRichiesta
11     GROUP BY mdg.data, mdg.pranzo_cena, m.nome, p.id, p.nome, p.descrizione
12     ORDER BY mdg.pranzo_cena, p.id;
13 END
```

2.7.2 aggiungi_menu

Questa procedura viene utilizzata per aggiungere un nuovo menu al database ed associargli una lista piatti. Vengono passate a parametro il nome del menu ed una stringa (p_piatti_csv) che contiene la lista piatti.
Successivamente viene inserita una tupla in menu con il nome di p_nome. L'id del menu viene settato con un `LAST_INSERT_ID`.
La parte fondamentale di questa procedura è il ciclo `WHILE` dove si aggiungono i piatti al menù.
In ordine avvengono le seguenti operazioni:

1. **Inizializzazione** `SET v_remaining = p_piatti_csv`.
2. **Ciclo** finché c'è testo rimanente all'interno i `v_remaining` si prosegue con le operazioni che seguono:
   (a) `LOCATE()` Cerca la prima virgola in `v_remaining`.
   (b) Se la trova allora viene recuperato tutto ciò che c'è prima della stringa e viene trasformato in un numero tramite il `CAST`. Poi viene assegnato alla variabile `v_piatto_id`.
   (c) Se non la trova allora siamo all'ultimo piatto da aggiungere al menu. Viene svolto un `CAST` dell'ultima parte di stringa.
   (d) L'ultima fase è un `INSERT` di `v_piatto_id` con il relativo `v_menu_id` nella tabella `piatti_menu`.

```
1  CREATE PROCEDURE `aggiungi_menu` (IN `p_nome` VARCHAR(20), IN `p_piatti_csv` TEXT)   BEGIN
2      DECLARE v_menu_id INT;
3      DECLARE v_piatto_id INT;
4      DECLARE v_pos INT;
5      DECLARE v_remaining TEXT;
6
7      DECLARE EXIT HANDLER FOR SQLEXCEPTION
8      BEGIN
9          ROLLBACK;
10         RESIGNAL;
11     END;
12
13     START TRANSACTION;
14
15     INSERT INTO menu (nome) VALUES (p_nome);
16     SET v_menu_id = LAST_INSERT_ID();
17
18     SET v_remaining = p_piatti_csv;
```

```
    WHILE LENGTH(v_remaining) > 0 DO
        SET v_pos = LOCATE(',', v_remaining);
        IF v_pos > 0 THEN
            SET v_piatto_id = CAST(SUBSTRING(v_remaining, 1, v_pos - 1) AS UNSIGNED);
            SET v_remaining = SUBSTRING(v_remaining, v_pos + 1);
        ELSE
            SET v_piatto_id = CAST(v_remaining AS UNSIGNED);
            SET v_remaining = '';
        END IF;

        INSERT INTO piatti_menu (id_menu, id_piatto) VALUES (v_menu_id, v_piatto_id);
    END WHILE;

    COMMIT;
END
```

2.7.3 get_menu_programmati

Procedura utilizzata per recuperare i menu programmati per le date future; trova impiego nel pannello operatori, nella sezione dedicata ai menu.

Alla procedura non vengono passati parametri, in quanto il punto di riferimento temporale per il calcolo è la data corrente. Nello specifico, viene eseguita un'operazione di `SELECT` per restituire i dati dei menu, filtrando i risultati tramite la condizione `WHERE DATE(mdg.data) >= CURDATE()`, in modo da recuperare esclusivamente i record con data odierna o successiva.

```
    CREATE PROCEDURE `get_menu_programmati` ()   BEGIN
    SELECT mdg.data, mdg.pranzo_cena, m.nome AS menu_nome
    FROM menudelgiorno mdg
    INNER JOIN menu m ON mdg.menuID = m.id
    WHERE DATE(mdg.data) >= CURDATE()
    ORDER BY mdg.data, mdg.pranzo_cena;
END
```

2.7.4 GetMenuDelGiorno

Questa procedura è impiegata per la visualizzazione del menu nell'area pubblica.
Riceve come parametro la data di riferimento per il menu da visualizzare.
Le operazioni eseguite sono le seguenti:

1. Selezione (`SELECT`) della data del menu.
2. Utilizzo del costrutto `CASE` per restituire la stringa 'Pranzo' se il tipo di pasto nel database è salvato come 0, altrimenti 'Cena'.
3. Selezione (`SELECT`) dei dati relativi al piatto.
4. Utilizzo di `GROUP_CONCAT` per recuperare tutti gli allergeni del piatto e unirli in un'unica stringa.
5. Esecuzione di `INNER JOIN` e `LEFT JOIN` per collegare le tabelle e recuperare le informazioni necessarie per il menu del giorno.
6. Applicazione della clausola `WHERE` per filtrare i risultati in base alla `dataRichiesta`.

```
1  CREATE PROCEDURE `GetMenuDelGiorno` (IN `dataRichiesta` DATE)   BEGIN
2      SELECT
3          mdg.data,
4          CASE
5              WHEN mdg.pranzo_cena = 0 THEN 'Pranzo'
6              ELSE 'Cena'
7          END AS tipo_pasto,
8          m.nome AS menu_nome,
9          p.id AS piatto_id,
10         p.nome AS piatto_nome,
11         p.descrizione AS piatto_descrizione,
12         GROUP_CONCAT(a.nome SEPARATOR ', ') AS allergeni
13     FROM menudelgiorno mdg
14     INNER JOIN menu m ON mdg.menuID = m.id
15     INNER JOIN piatti_menu pm ON m.id = pm.id_menu
16     INNER JOIN piatto p ON pm.id_piatto = p.id
17     LEFT JOIN piatti_allergeni pa ON p.id = pa.id_piatto
18     LEFT JOIN allergene a ON pa.id_allergene = a.id
19     WHERE DATE(mdg.data) = dataRichiesta
20     GROUP BY mdg.data, mdg.pranzo_cena, m.nome, p.id, p.nome, p.descrizione
21     ORDER BY mdg.pranzo_cena, p.id;
22 END
23
```

2.7.5 inserisci_consumazione

Questa procedura è utilizzata per registrare nuove consumazioni, sia a seguito dell'acquisto di un pasto nell'area studenti, sia in caso di aggiunta manuale tramite l'area operatori.

La procedura accetta come parametri la quantità di pasti da aggiungere, il codice fiscale e la fascia dello studente.

Le operazioni eseguite sono le seguenti:

1. Verifica dell'esistenza del codice fiscale (`cf`) dello studente tramite un'operazione di `COUNT(*)` sulla tabella `info_studente`.

2. Esecuzione di un ciclo `WHILE` che, fintanto che il contatore `i` è minore di `p_quantità`, effettua un `INSERT` nella tabella `pasto` per registrare la consumazione.

```
1  CREATE PROCEDURE `inserisci_consumazione` (IN `p_cf` VARCHAR(16), IN `p_fascia` INT, IN `p_quantit` INT)   BEGIN
2      DECLARE v_esiste INT;
3      DECLARE errore INT DEFAULT 0;
4      DECLARE i INT DEFAULT 0;
5      DECLARE EXIT HANDLER FOR SQLEXCEPTION
6          BEGIN ROLLBACK; END;
7
8         SELECT COUNT(*) INTO v_esiste FROM info_studente WHERE cf = p_cf;
9
10     IF v_esiste = 0 THEN
11         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Studente non trovato';
12     END IF;
13
14
15         START TRANSACTION;
16
17         WHILE i < p_quantit DO
18 INSERT INTO pasto (id, dataAcquisto, dataUtilizzo, studente, menuID, fascia)
19 VALUES (NULL, NOW(), NULL, p_cf, NULL, p_fascia);
20 SET i = i + 1;
21 END WHILE;
22 COMMIT;
23
```

2.8 Sistema generazione e validazione pasto

2.8.1 genera_token

La procedura `genera_token` riceve il codice fiscale dello studente e una chiave segreta. La logica si articola in tre fasi.

Per prima cosa verifica se esiste già un token attivo associato allo studente, ossia un record nella tabella `token` con `data_scadenza > NOW()`. In tal caso lo restituisce direttamente senza crearne uno nuovo.

In caso contrario, elimina i token scaduti dello studente dalla tabella `token` e cerca un pasto disponibile nella tabella `pasto`, ovvero un record con `dataUtilizzo IS NULL`. Se non ne trova, solleva un'eccezione con `SIGNAL SQLSTATE '45000'`.

Se invece un pasto è disponibile, calcola il token come `SHA2-256` della concatenazione di chiave segreta, codice fiscale e identificativo del pasto, lo inserisce in `token` con una scadenza di 5 minuti (`DATE_ADD(NOW(), INTERVAL 5 MINUTE)`) e lo restituisce.

```
CREATE PROCEDURE `genera_token` (IN `p_studente` VARCHAR(16), IN `p_codice_segreto` VARCHAR(255))   BEGIN
    DECLARE v_token_esistente VARCHAR(255);
    DECLARE v_pasto_esistente INT;
    DECLARE v_scadenza_esistente TIMESTAMP;
    DECLARE v_pasto_id INT;
    DECLARE v_token_id VARCHAR(255);

    -- Controlla se esiste già un token attivo (non scaduto)
    SELECT id, pasto, data_scadenza
    INTO v_token_esistente, v_pasto_esistente, v_scadenza_esistente
    FROM token
    WHERE studente = p_studente
    AND data_scadenza > NOW()
    LIMIT 1;
    IF v_token_esistente IS NOT NULL THEN
        -- Restituisci il token esistente
        SELECT
            v_token_esistente AS token,
            v_pasto_esistente AS pasto,
            v_scadenza_esistente AS scadenza,
            1 AS esistente;
    ELSE
        -- Elimina eventuali token scaduti dello studente
        DELETE FROM token
        WHERE studente = p_studente
        AND data_scadenza <= NOW();
```

```
        -- Cerca un pasto disponibile (non ancora utilizzato)
        SELECT id INTO v_pasto_id
        FROM pasto
        WHERE studente = p_studente
        AND dataUtilizzo IS NULL
        LIMIT 1;
        IF v_pasto_id IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Lo studente non ha pasti disponibili';
        ELSE
            -- Genera il token hashando codicesegreto+studente+pasto
            SET v_token_id = SHA2(CONCAT(p_codice_segreto, p_studente, v_pasto_id), 256);

            -- Inserisci il token
            INSERT INTO token (id, studente, pasto, data_scadenza)
            VALUES (v_token_id, p_studente, v_pasto_id, DATE_ADD(NOW(), INTERVAL 5 MINUTE));

            -- Restituisci il token creato
            SELECT
                v_token_id AS token,
                v_pasto_id AS pasto,
                DATE_ADD(NOW(), INTERVAL 5 MINUTE) AS scadenza,
                0 AS esistente;
        END IF;
    END IF;
END
```

2.8.2 valida_token

La procedura `valida_token` riceve l'identificativo del token scansionato e ne verifica la validità eseguendo una serie di controlli in sequenza.

Per prima cosa cerca il token nella tabella `token`: se non esiste solleva un'eccezione. Se esiste ma risulta scaduto (`data_scadenza < NOW()`), lo elimina e solleva un'eccezione. Infine controlla che il pasto associato non sia già stato utilizzato (`dataUtilizzo IS NOT NULL` in `pasto`): in caso affermativo elimina il token e solleva un'eccezione.

Superati tutti i controlli, determina il tipo di pasto in base all'ora corrente: pranzo se l'orario è compreso tra le 11:00 e le 15:00, cena tra le 18:00 e le 22:00. Utilizza questa informazione per ricercare il menu del giorno nella tabella `menudelgiorno` corrispondente alla data odierna e al tipo di pasto.

Infine aggiorna il record in `pasto` impostando `dataUtilizzo = CURDATE()` e associando il menu trovato (o `NULL` se non programmato), ed elimina il token ormai consumato dalla tabella `token`.

```
    DECLARE v_pasto_id INT;
    DECLARE v_studente VARCHAR(16);
    DECLARE v_data_scadenza TIMESTAMP;
    DECLARE v_menu_id INT DEFAULT NULL;
    DECLARE v_pranzo_cena TINYINT;
    DECLARE v_ora_corrente TIME;

    -- Cerca il token
    SELECT pasto, studente, data_scadenza
    INTO v_pasto_id, v_studente, v_data_scadenza
    FROM token
    WHERE id = p_token;

    -- Se il token non esiste
    IF v_pasto_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Token non valido';
    END IF;

    -- Se il token è scaduto
    IF v_data_scadenza < NOW() THEN
        -- Elimina il token scaduto
        DELETE FROM token WHERE id = p_token;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Token scaduto';
    END IF;

    -- Controlla se il pasto è già stato utilizzato
    IF (SELECT dataUtilizzo FROM pasto WHERE id = v_pasto_id) IS NOT NULL THEN
        DELETE FROM token WHERE id = p_token;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Pasto già utilizzato';
    END IF;

    -- Determina se è pranzo (0) o cena (1) in base all'orario
    SET v_ora_corrente = CURRENT_TIME();
    IF v_ora_corrente BETWEEN '11:00:00' AND '15:00:00' THEN
        SET v_pranzo_cena = 0; -- Pranzo
    ELSEIF v_ora_corrente BETWEEN '18:00:00' AND '22:00:00' THEN
        SET v_pranzo_cena = 1; -- Cena
    END IF;

    -- Cerca il menu del giorno (se esiste)
    SELECT menuID INTO v_menu_id
    FROM menudelgiorno
    WHERE DATE(data) = CURDATE() AND pranzo_cena = v_pranzo_cena
    LIMIT 1;
    -- Aggiorna il pasto con data utilizzo e menu
    UPDATE pasto
    SET dataUtilizzo = CURDATE(),
        menuID = v_menu_id
    WHERE id = v_pasto_id;
```

```
    -- Elimina il token usato
    DELETE FROM token WHERE id = p_token;

    -- Restituisci successo con info
    SELECT
        'OK' AS esito,
        v_studente AS studente,
        v_pasto_id AS pasto,
        CASE v_pranzo_cena WHEN 0 THEN 'Pranzo' ELSE 'Cena' END AS tipo_pasto,
        v_menu_id AS menu_id,
        (SELECT CONCAT(nome, ' ', cognome) FROM info_studente WHERE cf = v_studente) AS nome_studente;

END
```

2.9 Trigger

```

CREATE TRIGGER `decrementa_pasti` AFTER UPDATE ON `pasto` FOR EACH ROW BEGIN
        IF new.dataUtilizzo IS NOT NULL THEN
                UPDATE info_studente SET pasti = pasti-1
                WHERE cf = new.studente;
        END IF;
END
```

Listing 1: Trigger: decrementa_pasti

```

CREATE TRIGGER `decrementa_pasti_cancellazione` AFTER DELETE ON `pasto` FOR EACH ROW BEGIN
   IF OLD.dataUtilizzo IS NULL THEN
        UPDATE info_studente
        SET pasti = pasti - 1
        WHERE cf = OLD.studente;
    END IF;
END
```

Listing 2: Trigger: decrementa_pasti_cancellazione

```sql
CREATE TRIGGER `incrementa_pasti` AFTER INSERT ON `pasto` FOR EACH ROW BEGIN
    UPDATE info_studente
    SET pasti = pasti + 1
    WHERE cf = NEW.studente;
END
```

Listing 3: Trigger: incrementa_pasti

3 Fonti

Riferimenti bibliografici

[1] D. Shim, *qrcodejs*, libreria JavaScript per la generazione di codici QR, GitHub. github.com/davidshimjs/qrcodejs

[2] Mebjas, *html5-qrcode*, libreria JavaScript per la scansione di codici QR, GitHub. github.com/mebjas/html5-qrcode

[3] Repo del progetto https://github.com/DanieleSimula/ProgettoMensa

4 Limitazioni

- Nella sezione operatore non si possono aggiungere gli allergeni ai piatti.
- Nella sezione operatore si possono rimuovere solo un pasto alla volta.
- Gli studenti e gli admin non possono essere aggiunti o eliminati tramite interfaccia.
- Non si possono creare nuove allergeni.