🔪 Esercitazione 1 — Gestione Rubrica Contatti (CRUD con PDO)

Obiettivo: Realizzare una porzione di web app che consenta la gestione di una rubrica di contatti, utilizzando form HTML, sessioni PHP e PDO per il collegamento a un database MySQL.

Requisiti funzionali:

- Login con sessione PHP (mock, hardcoded: utente = admin, password = 1234).
- Form per **aggiungere un nuovo contatto**: nome, cognome, email, telefono.
- Visualizzazione in tabella dei contatti presenti nel database.
- Funzionalità di **modifica** e **cancellazione** contatto via form.
- Gestione degli errori e messaggi di successo tramite sessione.

Requisiti tecnici:

- Utilizzare PDO per connettersi al database.
- I dati devono essere salvati in una tabella `contatti` con chiave primaria auto-incrementale.
- Sanificazione dell'input (uso di `htmlspecialchars` e prepared statements).

🔪 Esercitazione 1 – Rubrica Contatti (CRUD con PDO)

Struttura del database (`rubrica.sql`)

```sql
CREATE TABLE contatti (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    telefono VARCHAR(20) NOT NULL
);
```

Dati mock

```sql
INSERT INTO contatti (nome, cognome, email, telefono) VALUES
('Mario', 'Rossi', 'mario.rossi@example.com', '3331234567'),
('Luca', 'Bianchi', 'luca.bianchi@example.com', '3349876543'),
('Anna', 'Verdi', 'anna.verdi@example.com', '3351122334');
```

🔪 Esercitazione 2 — Diario di Bordo (Lettura e Scrittura File)

Obiettivo: Creare un'applicazione che consenta all'utente autenticato di scrivere e leggere pagine di diario salvate come file di testo, organizzate per data.

Requisiti funzionali:

- Login con sessione PHP (mock, hardcoded).
- Form per **scrivere una nuova voce di diario**: data (campo `date`) e testo.
- Il file deve essere salvato come `YYYY-MM-DD.txt` in una directory `diario/`.
- Visualizzazione elenco voci di diario con possibilità di cliccare e leggerne il contenuto.
- Visualizzazione messaggi di conferma/successo tramite sessione.

Requisiti tecnici:

- Lettura e scrittura dei file con funzioni come `file_put_contents()` e `file_get_contents()`.
- Creazione della directory `diario/` se non esistente.
- Sanitizzazione input utente.
- Gestione degli errori (es. file non accessibile).

🪛 Esercitazione 2 – Diario di Bordo (File)

File da leggere

Salvato come: `diario/2025-06-30.txt`

Contenuto del file

```
Giornata intensa. Ho completato l'esercizio sul CRUD con PDO e ho iniziato
quello sul diario.
La gestione dei file con PHP è più semplice del previsto, ma bisogna stare
attenti ai permessi.
Domani mi dedico all'upload!
```

🪛 Esercitazione 3 — Gestione Immagini (Upload e Galleria)

Obiettivo: Creare una galleria personale per l'utente autenticato, con possibilità di caricare immagini via form e visualizzarle in una griglia.

Requisiti funzionali:

- Login con sessione PHP.
- Form HTML per **caricare un'immagine** (`<input type="file">`).
- Le immagini devono essere salvate nella cartella `upload/` con nome univoco.
- Visualizzazione in pagina di tutte le immagini caricate (galleria con thumbnail).
- Validazione tipo e dimensione file.
- Messaggi di errore/successo in sessione.

Requisiti tecnici:

- Uso di `move_uploaded_file()` e `$_FILES`.
- Limitazione a file JPEG/PNG con max 2 MB.
- Creazione automatica della directory `upload/` se non esistente.
- Uso di `session_start()` e messaggi flash in sessione.

🔪 Esercitazione 3 – Upload Immagini (Galleria)

Questa esercitazione **non richiede un database**, ma puoi opzionalmente usarlo per tenere traccia delle immagini caricate (es. nome file e data upload).

Struttura opzionale (`upload_gallery.sql`)

```
CREATE TABLE immagini (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome_file VARCHAR(255) NOT NULL,
    data_upload DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Dati mock

```
INSERT INTO immagini (nome_file) VALUES
('gatto.jpg'),
('tramonto.png'),
('montagna.jpeg');
```

Puoi anche popolare manualmente la cartella `upload/` con immagini placeholder.

🔪 Esercitazione 4 — Servizio REST per Task Manager (CRUD API)

Obiettivo: Realizzare un servizio RESTful in PHP che consenta la gestione di una lista di task (to-do list), con risposta in formato JSON.

Requisiti funzionali:

- Lato **frontend**: pagina HTML con form per aggiungere un task (titolo e descrizione), visualizzazione elenco e pulsanti "completa" e "elimina".
- Lato **backend**: un endpoint `api/tasks.php` che accetta richieste HTTP (GET, POST, PUT, DELETE) e restituisce JSON.
- Sessione PHP usata per mantenere l'utente loggato e per validare accesso alle API.

Requisiti tecnici:

- CRUD implementato usando `$_SERVER['REQUEST_METHOD']`.
- Dati salvati in tabella MySQL `tasks` con `id`, `titolo`, `descrizione`, `completato`.
- Le risposte JSON devono seguire un formato standard: `{ "status": "ok", "data": ... }` oppure `{ "status": "error", "message": ... }`.
- Sanitizzazione input e gestione errori.

🔪 Esercitazione 4 – Task Manager (REST API)

Struttura del database (`taskmanager.sql`)

```sql
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titolo VARCHAR(100) NOT NULL,
    descrizione TEXT,
    completato BOOLEAN DEFAULT FALSE
);
```

Dati mock

```sql
INSERT INTO tasks (titolo, descrizione, completato) VALUES
('Preparare la presentazione', 'Slide per il corso di PHP avanzato',
FALSE),
('Scrivere documentazione API', 'Endpoint REST per il task manager', TRUE),
('Controllare esercizi studenti', 'Verifica esercitazione su file e
sessioni', FALSE);
```
