# Relazione Didattica: Progetto "Turing NEWS"

**Progetto realizzato da:**
**Maisano Tommaso** (Matricola: 544416)
**Salmeri Tindaro** (Matricola: 546941)

**Anno Accademico: 2024/2025**

---

## 1. Introduzione

Questo documento descrive l'architettura e le funzionalità del progetto "Turing NEWS", un'applicazione web concepita come una piattaforma social. Gli utenti possono registrarsi, effettuare il login, pubblicare post(composti da testo e un'immagine opzionale), commentare quelli altrui e aggiungere "like" a quelli di loro interesse.

L'obiettivo del progetto è stato quello di realizzare un'applicazione web semplice, separando nettamente la logica di frontend (gestione dell'interfaccia utente) da quella di backend (gestione dei dati e della logica) attraverso un'API RESTful.

---

## 2. Architettura e Tecnologie Utilizzate

Il sistema segue un'architettura **Client-Server** disaccoppiata.

- **Frontend (Client):** Realizzato in **HTML5**, **CSS3** e **JavaScript (ES6+)**. Non viene utilizzato nessun framework frontend (come React o Vue), ma il codice è strutturato in moduli JavaScript che gestiscono le chiamate API e la manipolazione dinamica del DOM.
- **Backend (Server):** Implementato come un'API RESTful in **PHP**. Il backend gestisce tutte le richieste del client, interagisce con il database e restituisce i dati in formato **JSON**.
- **Database:** Il database utilizzato è **MySQL**. La struttura del database è definita nel file di dump SQL e include tabelle per utenti, post, commenti e like, collegate tra loro da relazioni (foreign key).

La comunicazione tra client e server avviene tramite richieste HTTP asincrone (utilizzando `fetch` nel frontend) che interrogano gli endpoint PHP definiti nel backend.

---

## 3. Analisi del Database

1. `users`: Memorizza i dati degli utenti registrati, inclusi `username`, `email` e `password` (memorizzata come hash).

2. `posts`: Contiene le notizie pubblicate. Ogni post è collegato a un utente tramite la foreign key `user_id` e include `title`, `content` e un `image_url` opzionale.
3. `comments`: Memorizza i commenti degli utenti ai post. È una tabella di relazione N:M "virtuale" (molti commenti per post, molti commenti per utente) collegata sia a `posts` (`post_id`) sia a `users` (`user_id`).
4. `likes`: Gestisce i "mi piace". Una riga in questa tabella rappresenta un "like" di un `user_id` a un `post_id`. La chiave unica `unique_like (post_id, user_id)` garantisce che un utente possa mettere "like" a un post una sola volta.

Tutte le tabelle utilizzano `ON DELETE CASCADE` per le foreign key, garantendo che l'eliminazione di un utente o di un post rimuova automaticamente tutti i commenti e i like associati, mantenendo l'integrità referenziale.

---

## 4. Funzionalità del Backend (API)

Il backend è costituito da una serie di script PHP che fungono da endpoint API. Il file `api/config.php` è centrale: gestisce la connessione al database tramite **PDO**, imposta gli header **CORS** (permettendo al frontend di effettuare chiamate) e fornisce funzioni di utilità come `jsonResponse` e un sistema di autenticazione.

L'autenticazione è gestita tramite un token personalizzato: * Alla registrazione (`api/auth.php`) o al login, l'API genera un token (in questo caso, un `base64_encode` dell'ID utente e un timestamp) che viene inviato al client. * Per le rotte protette, il client deve inviare questo token nell'header `Authorization`. La funzione `validateToken()` sul server lo decodifica per identificare l'utente.

Gli endpoint principali sono:

### `api/auth.php`

- `POST (action: 'register')`: Registra un nuovo utente, previa validazione (email unica, username unico) e hashing della password tramite `password_hash()`.
- `POST (action: 'login')`: Verifica le credenziali dell'utente (email e `password_verify()`) e restituisce il token.
- `GET`: Verifica la validità del token inviato e restituisce i dati dell'utente (usato per `getCurrentUser()` nel frontend).

### `api/posts.php`

- `GET`: Recupera un elenco paginato di post. Supporta la paginazione (`page`, `per_page`), la ricerca testuale (`search` su titolo e contenuto) e il filtraggio

per utente (`user_id`). Calcola anche il numero di like e commenti tramite subquery.
- `POST`: Crea un nuovo post, associandolo all'utente autenticato (identificato tramite `validateToken()`).
- `DELETE`: Elimina un post. Crucialmente, esegue un controllo per assicurarsi che l'ID dell'utente (dal token) corrisponda al `user_id` del post che si sta tentando di eliminare.

#### `api/interactions.php`

- `POST (action: 'like')`: Implementa la logica "toggle" per i like. Se l'utente ha già messo like, lo rimuove; altrimenti, lo inserisce.
- `POST (action: 'comment')`: Aggiunge un nuovo commento a un post.
- `GET (action: 'comments')`: Restituisce tutti i commenti per un dato `post_id`.
- `GET (action: 'liked')`: Restituisce l'elenco dei post a cui l'utente autenticato ha messo "like" (usato nella Dashboard).
- `GET (action: 'check_like')`: Verifica se l'utente autenticato ha messo "like" a un post specifico.

#### `api/upload.php`

- `POST`: Gestisce il caricamento di file immagine (validando tipo MIME e dimensione). Genera un nome file univoco (combinando ID utente e timestamp) e salva l'immagine nella cartella `/uploads/`, restituendo l'URL relativo al client.

---

## 5. Funzionalità del Frontend (Client)

Il frontend è responsabile dell'interfaccia utente e dell'orchestrazione delle chiamate API.

#### `js/api-client.js`

Questo è il file più importante del frontend. Definisce una classe `ApiClient` che astrae tutte le chiamate `fetch` al backend. Questa classe gestisce automaticamente: * L'aggiunta dell'URL base (`API_BASE_URL`). * La gestione del token di autenticazione, prelevandolo da `localStorage` e inserendolo nell'header `Authorization` di ogni richiesta. * La serializzazione dei dati in JSON. * Fornisce metodi chiari (es. `api.login()`, `api.getPosts()`, `api.deletePost()`) che il resto dell'applicazione può usare senza conoscere i dettagli dell'implementazione `fetch`.

`js/auth-new.js`

Utilizza `api-client.js` per gestire lo stato dell'utente. Le funzioni `login()` e `register()` salvano l'utente in `localStorage`. `getCurrentUser()` controlla `localStorage` e verifica la validità del token con l'API. `checkAuthStatus()` aggiorna dinamicamente la UI (es. mostrando/nascondendo i link "Login" o "Logout").

`js/posts-new.js`

Contiene la logica per la visualizzazione e l'interazione con i post. * `renderPosts()` e `renderPaginatedPosts()`: Recuperano i post tramite `api.getPosts()` e generano dinamicamente l'HTML per le "news-card". * `setupPostInteractions()`: Aggiunge gli event listener ai pulsanti di ogni post per gestire like, commenti (mostrando/nascondendo la sezione) ed eliminazione. Il pulsante "Elimina" viene aggiunto dinamicamente solo se l'ID dell'utente loggato (`user.id`) corrisponde al `post.user_id`. * `setupPagination()` e `setupSearch()`: Gestiscono la paginazione e la barra di ricerca.

**Pagine HTML**

- `index.html`: La home page. Carica e visualizza il feed principale di tutte le notizie.
- `login.html` e `register.html`: Semplici form che utilizzano `auth-new.js` per eseguire l'autenticazione.
- `post.html`: Il form per la creazione di una nuova notizia. Gestisce l'upload dell'immagine (con anteprima) e, al submit, prima esegue l'upload dell'immagine (`api.uploadImage`) e poi, ottenuto l'URL, crea il post (`api.createPost`) con tutti i dati.
- `dashboard.html`: La pagina personale dell'utente. Utilizza `api.getUserPosts()` e `api.getLikedPosts()` per mostrare due sezioni separate: "Le tue notizie" e "Notizie che hai apprezzato".

---

## 6. Conclusione

Il nostro progetto "Turing NEWS" implementa un'applicazione web completa, basata su un'architettura disaccoppiata che separa chiaramente backend e frontend. Il backend API (PHP) gestisce in modo robusto l'autenticazione, la logica e l'accesso ai dati (MySQL), mentre il frontend (JavaScript) offre un'interfaccia utente dinamica e reattiva, orchestrando le chiamate API tramite un client centralizzato.