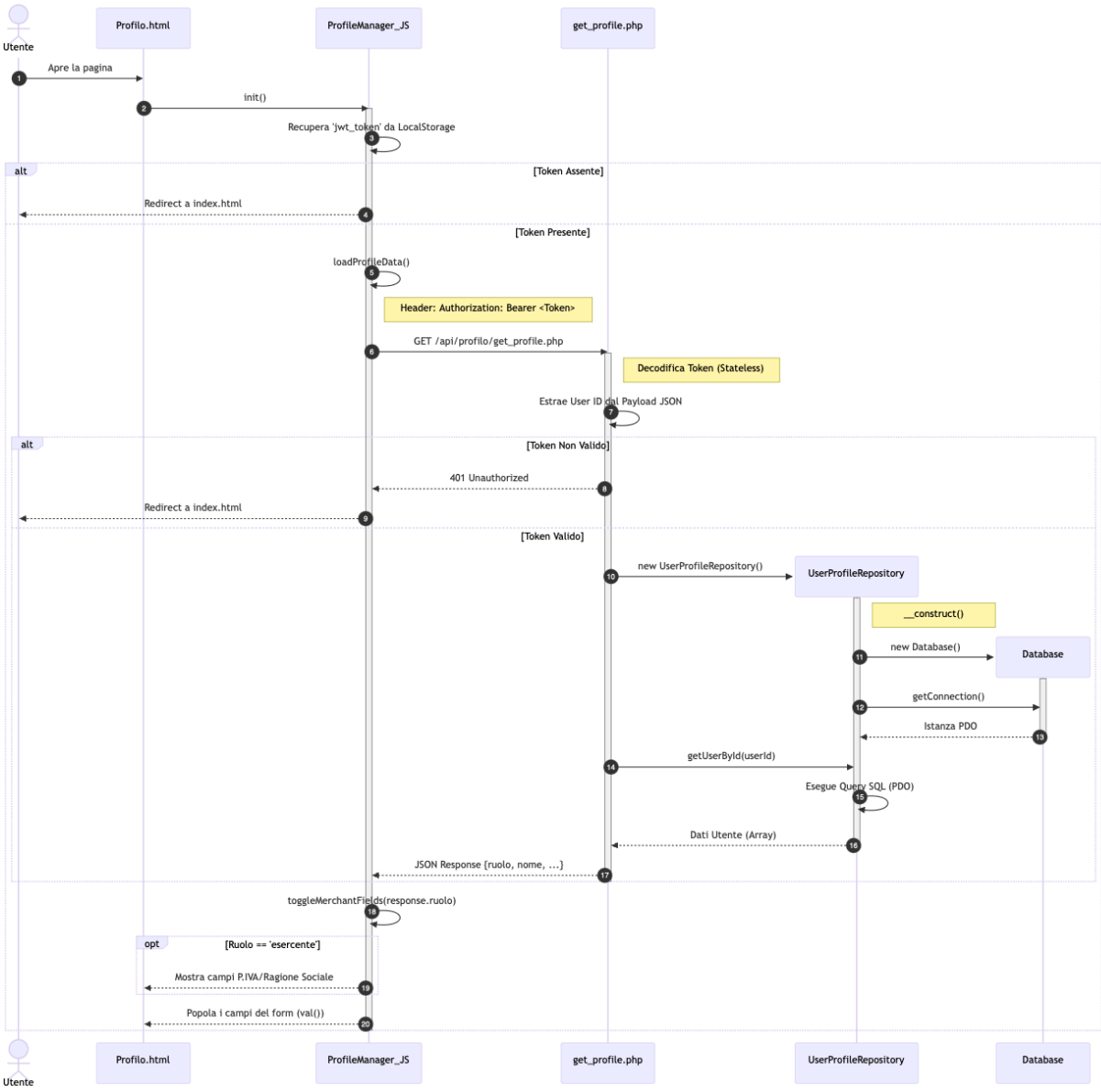
SVOLTO DA COLUCCI PASQUALE, MATR: 358141

Diagramma di Sequenza: Visualizzazione del Profilo, Modulo Gestione Profilo



Descrizione: Visualizzazione del Profilo, Modulo Gestione Profilo

Il diagramma di sequenza illustra il flusso dinamico delle interazioni necessarie per recuperare e visualizzare le informazioni dell'utente all'apertura della pagina del profilo. Il processo si concentra sulla comunicazione sicura tra il layer di presentazione client-side e la logica di business server-side, evidenziando l'adozione del protocollo **Stateless**.

1. Attori e Partecipanti

- **Utente:** L'attore che avvia l'interazione accedendo alla pagina.
- **View (Profilo.html):** La pagina HTML che ospita il form e i campi dati.
- **JS (ProfileManager_JS):** Il controller lato client responsabile della logica di visualizzazione, della gestione del Token e delle chiamate AJAX.
- **API (get_profile.php):** L'endpoint REST lato server che riceve la richiesta e orchestra il recupero dei dati.
- **Repo (UserProfileRepository):** Il componente Model che incapsula la logica di accesso ai dati.
- **DB (Database):** La classe wrapper per la connessione al database (PDO).

2. Flusso degli Eventi

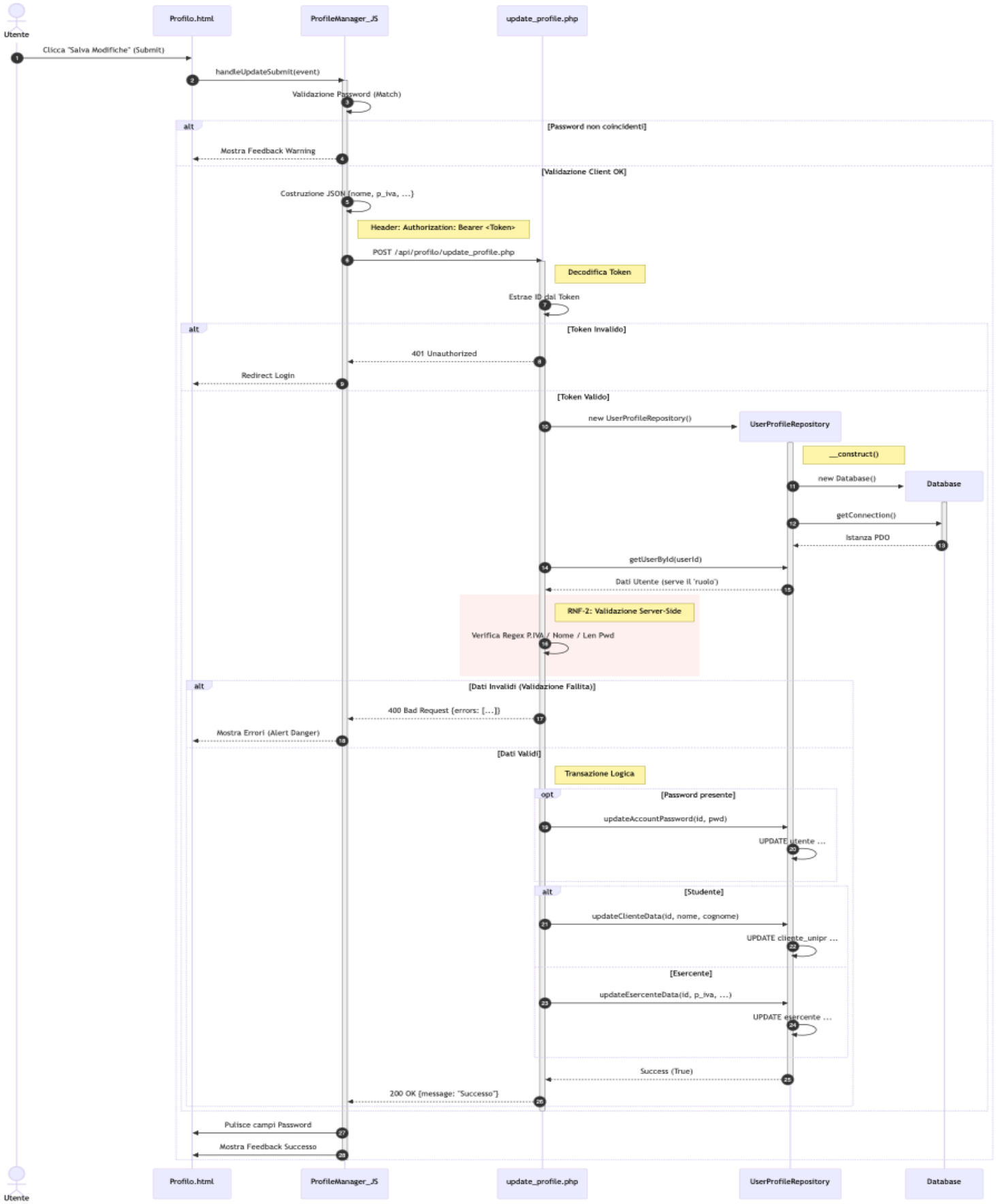
1. **Inizializzazione e Controllo Sicurezza:** All'apertura della pagina, la View invoca il metodo `init()` del `ProfileManager_JS`. Il sistema esegue immediatamente un controllo di sicurezza client-side: verifica la presenza del **Token JWT** nel `LocalStorage`.
    - *Caso Token Assente:* L'utente viene reindirizzato istantaneamente alla pagina di Login (`index.html`).
    - *Caso Token Presente:* Il flusso prosegue con il caricamento dei dati.
2. **Richiesta Dati (Stateless):** Il metodo `loadProfileData()` invia una richiesta asincrona (AJAX) di tipo GET all'endpoint `get_profile.php`. Invece, il Token viene inserito nell'Header HTTP `Authorization: Bearer <Token>`, garantendo che l'identità sia protetta e verificata dal server.
3. **Elaborazione Backend:** L'API riceve la richiesta e applica la logica **Stateless**:
    - Decodifica il payload del Token per estrarre l'ID utente (`extractUserIDFromToken`).
    - Se il token è invalido o scaduto, restituisce un errore `401 Unauthorized` che provoca il redirect al login lato client.
4. **Accesso ai Dati (Persistenza):** Se l'autenticazione ha successo, l'API istanzia dinamicamente il `UserProfileRepository`.
    - Il Repository, nel suo costruttore, crea una **nuova istanza** della classe `Database` (`new Database()`) e richiede una connessione PDO. Questo approccio garantisce l'isolamento della richiesta.
    - Viene eseguito il metodo `getUserById(userId)`, che interroga le tabelle coinvolte (UTENTE in join con CLIENTE_UNIPR o ESERCENTE).

5. **Risposta e Adattamento UI:** Il server restituisce un oggetto JSON contenente i dati anagrafici e il ruolo dell'utente. Il `ProfileManager_JS` riceve la risposta e invoca `toggleMerchantFields(ruolo)`:
    - Se il ruolo è **Studente**, vengono mostrati i campi Nome, Cognome e Matricola.
    - Se il ruolo è **Esercente**, l'interfaccia si adatta dinamicamente mostrando Ragione Sociale, P.IVA e Indirizzo di Ritiro. Infine, i valori vengono popolati nei campi del form HTML.

3. Note Tecniche

- **Autenticazione Stateless :** Il sistema non utilizza sessioni server-side. L'identificazione dell'utente avviene esclusivamente tramite la decodifica per-request del Token JWT.
- **Gestione Risorse:** La classe `Database` non è implementata come Singleton globale, ma viene istanziata puntualmente all'interno del Repository. Questo favorisce un ciclo di vita della connessione strettamente legato alla durata della singola richiesta HTTP.
- **Interfaccia Adattiva:** La logica di visualizzazione è *data-driven*: è la risposta del backend a determinare quali campi mostrare all'utente, permettendo di utilizzare un'unica pagina HTML per profilazioni diverse.

Diagramma di Sequenza: Aggiornamento del Profilo, Modulo Gestione Profilo



Descrizione: Aggiornamento del Profilo, Modulo Gestione Profilo

Il diagramma di sequenza descrive il processo di modifica dei dati utente, ponendo l'accento sulla sicurezza, sulla validazione ridondante e sulla gestione transazionale delle modifiche.

1. Attori e Partecipanti

- **Utente:** L'attore che immette i nuovi dati e conferma l'operazione.
- **View (Profilo.html):** L'interfaccia che raccoglie l'input (form HTML).
- **JS (ProfileManager_JS):** Il controller client-side che gestisce la pre-validazione e la comunicazione asincrona.
- **API (update_profile.php):** Il controller server-side che implementa la logica di validazione (RNF-2) e orchestrazione.
- **Repo (UserProfileRepository):** Il componente Model responsabile delle query SQL di aggiornamento.
- **DB (Database):** La classe wrapper per la connessione al database.

2. Flusso degli Eventi

1. **Input e Validazione Client:** L'utente clicca su "Salva Modifiche". Il `ProfileManager_JS` intercetta l'evento e performa una prima validazione sintattica (es. verifica che i campi "Password" e "Conferma Password" coincidano).
    - *Caso Errore:* L'invio viene bloccato e viene mostrato un avviso all'utente.
2. **Invio Richiesta Sicura:** Se il controllo passa, il client costruisce un payload JSON con i soli dati modificati. La richiesta viene inviata via POST includendo l'header `Authorization: Bearer <Token>`, senza esporre l'ID utente nell'URL o nel corpo della richiesta.
3. **Inizializzazione Backend:** L'API riceve la richiesta e:
    - Decodifica il Token per identificare l'utente (Stateless).
    - Istanzia il `UserProfileRepository` (e di conseguenza il `Database`) per recuperare il ruolo dell'utente corrente dal DB. Questo passaggio è necessario per sapere quali regole di validazione applicare.
4. **Validazione Ridondante:** Prima di scrivere nel database, l'API esegue una validazione rigorosa sui dati in ingresso (`validateInputData` inline nel codice PHP):
    - Controlla la lunghezza minima della password (se presente).
    - Verifica tramite Regex il formato della P.IVA (11 cifre) o del Nome/Cognome (solo lettere).
    - *Scenario Fallimento:* Se i dati sono malformati, l'API risponde con `400 Bad Request` e una lista di errori, proteggendo il database da input inconsistenti.
5. **Persistenza dei Dati:** Se la validazione ha successo, l'API invoca i metodi di aggiornamento specifici del repository:
    - `updateAccountPassword()` per la tabella `utente`.
    - `updateClienteData()` o `updateEsercenteData()` in base al ruolo. Le operazioni vengono eseguite sequenzialmente e l'esito viene restituito al client.

6. **Feedback:** Il client riceve la risposta `200 OK`, pulisce i campi sensibili (password) e mostra un messaggio di successo.

3. Note Tecniche

- **Difesa in Profondità:** Nonostante la validazione HTML/JS, il Backend riesegue tutti i controlli di integrità. Questo previene attacchi che bypassano il client (es. tramite Postman o cURL).
- **Gestione Dinamica delle Risorse:** Le classi di accesso ai dati (`Repository` e `Database`) vengono create "just-in-time" e distrutte al termine della richiesta, ottimizzando l'uso della memoria del server.
- **Separazione delle Responsabilità:** L'API funge da validatore e orchestratore, mentre il Repository si occupa esclusivamente dell'esecuzione delle query SQL, mantenendo il codice modulare e manutenibile.