**Documento Tecnico – Pagina di Login**

**1. Architettura e Struttura del Progetto**

**Tecnologia di Base:**

* **Framework:** Angular (solo front-end).
* **Gestione dello Stato:** Utilizzo di Angular Reactive Forms per la gestione degli input e delle validazioni.
* **Routing:** Configurazione del router Angular per il passaggio dalla pagina di login alla Home Page.

**2. Componenti e Struttura del Codice**

**Componenti Principali:**

* **LoginComponent:**
  + **Responsabilità:**
    - Visualizzare il form di login.
    - Gestire la validazione degli input.
    - Gestire l’interazione con il pulsante “Accedi”.
  + **Struttura Interna:**
    - Template HTML per la form.
    - CSS/SCSS per applicare lo stile (azzurro #428fdd e bianco).
    - File TypeScript per la logica di validazione e gestione degli eventi.
* **AuthService (Mock):**
  + **Responsabilità:**
    - Simulare l’autenticazione degli utenti.
    - Fornire metodi per la verifica delle credenziali.
  + **Nota:**
    - Non verrà implementata una vera comunicazione con un backend, ma verrà simulata una risposta positiva o negativa in base ai dati inseriti.

**3. Struttura del Layout e della UI**

**Layout della Pagina:**

* **Header (Opzionale):**
  + Includere il logo “Parthenope Banking” in alto, che rafforza il brand.
* **Form di Login:**
  + Posizionato centralmente, con una larghezza adeguata per garantire una buona usabilità sia su desktop che su mobile.
  + Utilizzo di container e card per separare visivamente il form dal background.

**Stile e Design:**

* **Colori:**
  + Azzurro (#428fdd) per pulsanti, bordi, e link.
  + Bianco per lo sfondo dei container e delle card.
* **Tipografia:**
  + Font moderni e leggibili, con dimensioni e pesi che garantiscono chiarezza.
* **Responsive Design:**
  + Utilizzo di CSS Grid o Flexbox per l’adattamento a diverse dimensioni di schermo.
  + Media query per ottimizzare l’esperienza su dispositivi mobili.

**4. Validazioni e Gestione degli Errori**

**Implementazione delle Validazioni:**

* **Reactive Forms:**
  + Definire un form con controlli per username/email e password.
  + Utilizzo dei validatori integrati di Angular (es. Validators.required, Validators.email).
* **Feedback Visivo:**
  + Messaggi di errore sotto ogni campo in caso di validazione fallita.
  + Disabilitazione del pulsante “Accedi” fino a quando tutti i campi obbligatori non sono validi.
* **Gestione degli Errori di Autenticazione:**
  + In caso di credenziali non riconosciute, visualizzazione di un messaggio generico tipo “Credenziali non valide” in una posizione prominente sopra o sotto il form.

**5. Routing e Flusso di Navigazione**

* **Routing:**
  + Configurazione di una rotta dedicata per la pagina di login (es. /login).
  + Dopo l’autenticazione simulata, eseguire il redirect alla Home Page (es. /home).
* **Guardie di Rotta:**
  + Possibile utilizzo di guardie (route guards) per evitare l’accesso alle pagine interne se l’utente non è autenticato (anche se in questo contesto si tratta di una simulazione).

**6. Considerazioni sulla Sicurezza (Simulata)**

* **Indicazioni Visive:**
  + Mostrare simboli o messaggi che indichino una connessione sicura (es. icona di lucchetto accanto al form).
* **Gestione delle Credenziali:**
  + Anche se la verifica è simulata, strutturare il codice in modo da poter integrare facilmente meccanismi di sicurezza reali (es. HTTPS, token JWT) in una futura evoluzione.

**7. Scalabilità e Manutenibilità**

* **Modularità del Codice:**
  + Separazione chiara tra la logica di presentazione (componenti) e la logica di business (servizi).
  + Utilizzo di best practices Angular per mantenere il codice facilmente estendibile e testabile.
* **Future Integrazioni:**
  + La struttura scelta permette di passare facilmente a un’autenticazione reale integrando chiamate HTTP verso un backend, mantenendo intatta la logica di validazione e il flusso utente.