**Architettura Tecnica – Parthenope Banking**

**1. Introduzione**

"Parthenope Banking" è un’applicazione front-end sviluppata con Angular, progettata per simulare un sistema di internet banking. L’obiettivo principale è fornire un’interfaccia utente intuitiva e sicura per la gestione di operazioni bancarie simulate, come login, visualizzazione dei dettagli del conto e creazione di bonifici.

Questo documento descrive l’architettura tecnica dell’applicativo, elencando le tecnologie, le librerie e le dipendenze necessarie, oltre ai requisiti per l’installazione e il deployment su un ambiente di produzione.

**2. Architettura Tecnica Generale**

L’applicativo è strutturato come una Single Page Application (SPA) basata su Angular. Il front-end è organizzato in componenti modulari, che includono pagine di login, home page, dettagli conto e creazione di bonifici. Pur essendo una simulazione con dati mock, l’architettura è concepita per facilitare future integrazioni con backend reali e servizi di autenticazione sicura.

L’applicazione, essendo solo front-end, può essere distribuita su un qualsiasi web server statico, mentre lo sviluppo è supportato dall’utilizzo di strumenti moderni per il build e il deployment.

**3. Tecnologie e Dipendenze**

**3.1 Framework e Librerie Principali**

* **Angular:**
  + Utilizzato come framework principale per lo sviluppo dell’applicativo SPA.
  + Angular CLI per la gestione del ciclo di vita del progetto (creazione, build, testing, etc.).
* **Angular Material (o alternativa UI):**
  + Fornisce componenti UI predefiniti per una maggiore coerenza visiva e usabilità.
* **RxJS:**
  + Utilizzato per la gestione degli stream di dati e delle operazioni asincrone.
* **Bootstrap o CSS Grid/Flexbox:**
  + Opzionale per il supporto al responsive design e per una rapida definizione degli stili.

**3.2 Librerie e Strumenti Aggiuntivi**

* **TypeScript:**
  + Linguaggio principale per lo sviluppo dell’applicazione Angular, che consente una maggiore robustezza e tipizzazione del codice.
* **SCSS/CSS:**
  + Utilizzato per la definizione degli stili, garantendo una palette coerente (azzurro #428fdd e bianco).
* **Linter e Formattatori:**
  + ESLint, Prettier per garantire la qualità e la coerenza del codice.
* **Testing Frameworks:**
  + Jasmine e Karma per i test unitari.
  + Protractor (o Cypress) per i test end-to-end.

**4. Requisiti di Installazione e Ambiente**

**4.1 Requisiti di Sistema**

* **Node.js:**
  + Versione LTS raccomandata (es. Node.js 14 o superiore) per supportare Angular CLI e il build del progetto.
* **NPM o Yarn:**
  + Gestore di pacchetti per installare le dipendenze del progetto.
* **Ambiente di Sviluppo:**
  + Editor di codice (es. Visual Studio Code) configurato per lo sviluppo in TypeScript e Angular.

**4.2 Application Server / Web Server**

* **Web Server Statico:**
  + L’applicativo, una volta buildato, può essere distribuito tramite un server HTTP statico come NGINX o Apache HTTP Server.
  + In alternativa, può essere ospitato su piattaforme cloud come AWS S3 (con CloudFront), Netlify, o Vercel, che offrono hosting per SPA.
* **Configurazione del Server:**
  + Assicurare la corretta gestione delle rotte: in una SPA è necessario configurare il server per effettuare il fallback alla index.html in caso di richieste a URL non riconosciuti.

**4.3 Dipendenze di Sistema**

* **Sistema Operativo:**
  + Compatibile con Windows, macOS o Linux.
* **Ambiente di Build:**
  + Accesso a terminale/console per eseguire comandi npm/yarn.
* **Versionamento:**
  + Git per il controllo della versione del codice sorgente.

**5. Processo di Build e Deployment**

**5.1 Build dell’Applicazione**

* **Angular CLI Build:**
  + Utilizzo di ng build --prod per generare la versione ottimizzata e minificata dell’applicativo.
* **Ottimizzazione:**
  + Configurazione di cache busting e lazy loading per migliorare le performance.

**5.2 Deployment**

* **Distribuzione su Web Server:**
  + Copiare i file generati (cartella dist/) sul web server statico scelto (es. NGINX, Apache).
* **Configurazione HTTPS:**
  + Assicurare che il server utilizzi certificati SSL per garantire la sicurezza delle comunicazioni, soprattutto per l’accesso ad un’app di internet banking (anche se simulata).

**6. Integrazione con Altri Servizi (Opzionale)**

* **Servizi di Monitoraggio e Logging:**
  + Integrazione con strumenti come Sentry per il monitoraggio degli errori in produzione.
* **CDN:**
  + Utilizzo di una Content Delivery Network per distribuire i file statici e migliorare la latenza per gli utenti finali.
* **CI/CD Pipelines:**
  + Configurazione di pipeline (es. GitHub Actions, GitLab CI) per automatizzare build, test e deployment.

**7. Considerazioni su Sicurezza e Performance**

* **Sicurezza:**
  + Anche se l’applicativo è simulato, utilizzare best practice nella gestione delle dipendenze e nel deployment per prevenire vulnerabilità note.
  + Configurare il server per limitare attacchi comuni (es. XSS, CSRF) e utilizzare HTTPS.
* **Performance:**
  + Ottimizzazione del bundle di Angular tramite tree shaking e lazy loading.
  + Minificazione dei file CSS e JavaScript, compressione tramite GZIP o Brotli sul web server.

**8. Conclusioni**

Il documento di architettura tecnica fornisce una panoramica completa delle tecnologie e delle dipendenze necessarie per lo sviluppo, il build e il deployment dell’applicativo "Parthenope Banking". L’uso di Angular, insieme a librerie e strumenti moderni, garantisce un’applicazione modulare, scalabile e facilmente manutenibile. La configurazione di un ambiente di deployment su web server statico, integrato con misure di sicurezza e ottimizzazione delle performance, assicura che l’applicativo offra un’esperienza utente fluida e sicura, in linea con le aspettative di un sistema di internet banking.