🧠 Aspetti da tenere a mente (framework e backend)

1. **Architettura dei servizi**
   - L'onboarding è un microservizio indipendente che espone un'API REST (FastAPI) su :8001. Quest'API comunica con il backend CGS (servizio principale su :8000) per orchestrare la generazione di contenutiraw.githubusercontent.com.
   - Il progetto è organizzato in moduli: c'è una directory onboarding che contiene codice, modelli e documentazione per questo serviziogithub.com.
   - La configurazione CORS è aperta in sviluppo (allow_origins=["*"]) e va limitata ai domini di produzione prima del deployraw.githubusercontent.com.
2. **Endpoints principali**
   L'API espone un router con prefisso /api/v1/ onboardingraw.githubusercontent.com. I metodi principali sono:
   - POST /start: crea una sessione, avvia la ricerca tramite Perplexity e genera le domande di chiarimentoraw.githubusercontent.com. Restituisce un session_id, lo stato iniziale e un riepilogo dello snapshot con un messaggio per l'utenteraw.githubusercontent.com.
   - POST /{session_id}/answers: valida le risposte, costruisce il payload e lancia il workflow CGS; la risposta include eventuale anteprima del contenuto, titolo, metrica di costo, stato di delivery, ecc.raw.githubusercontent.com.
   - GET /{session_id} e GET /{session_id}/status: permettono di recuperare rispettivamente i dettagli completi della sessione o solo lo stato corrente, incluso se lo snapshot è presente, eventuale ID del run CGS e stato di deliveryraw.githubusercontent.com.
   - GET /health: endpoint per la verifica dello stato del servizio e dei sottosistemi (Perplexity, Gemini, CGS, Supabase, ecc.)raw.githubusercontent.com.
3. **Modello di stato**
   Una sessione può trovarsi in diversi stati: created, researching, synthesizing, awaiting_user, payload_ready, executing, delivering, done, failed. Il frontend deve gestire questi stati con un meccanismo di polling (3 s durante la ricerca/sintesi, 5 s durante l'esecuzione) e interrompere il polling quando si raggiunge uno stato finale (awaiting_user, done o failed).
4. **Validazione e tipologie di domanda**
   Ogni domanda ha un tipo atteso (string, enum, boolean, number). La validazione sul backend richiede che gli enum corrispondano esattamente alle opzioni fornite e che siano presenti tutte le risposte. In caso contrario l'API restituirà errori con messaggi espliciti.
5. **Sicurezza e configurazione**

- Utilizza variabili d'ambiente per impostare l'URL dell'API di onboarding (NEXT_PUBLIC_ONBOARDING_API_URL) e restringere i domini CORS in produzione.
- Il backend Onboarding fornisce log dettagliati ed eccezioni gestite globalmente, ma eventuali errori dovrebbero essere mostrati con gentilezza all'utente e registrati per debugging.

🛠️ Piano per realizzare il frontend (React/Next.js suggerito)

1. **Preparazione del progetto**
   - Crea un nuovo progetto React o Next.js con TypeScript (consigliato) e configura una libreria UI come Tailwind, Material-UI o Chakra.
   - Aggiungi un client HTTP (es. fetch nativo o axios) e, se necessario, una libreria per lo stato globale (es. React context o Zustand).
2. **API client**
   - Implementa una classe OnboardingAPI come da guida, con metodi statici start, getSession, submitAnswers, execute e healthCheck che incapsulano le chiamate fetch. Gestisci la serializzazione in JSON e propaga gli errori con messaggi chiari.
   - Configura l'URL base usando NEXT_PUBLIC_ONBOARDING_API_URL, così da poter cambiare ambiente senza toccare il codice.
3. **Gestione dello stato e polling**
   - Crea un hook useOnboarding che accetta un sessionId e restituisce lo stato della sessione, eventuali errori e una funzione refetch.
   - Implementa il polling nei soli stati asincroni (researching, synthesizing, executing, delivering), usando setInterval con intervalli differenti (3 s o 5 s) e cancellando l'intervallo quando lo stato diventa awaiting_user, done o failed.
4. **Componenti UI**
   - **OnboardingForm**: un form con campi per brand_name, website, goal ed user_email. Aggiungi validazioni sul client (min. 2 caratteri per il brand, URL ed email validi, goal obbligatorio). Al submit chiama OnboardingAPI.start e memorizza il session_id.
   - **LoadingResearch**: mostra una barra di progresso o checklist mentre l'API effettua ricerca e sintesi. Puoi utilizzare animazioni per rendere l'attesa meno noiosa.
   - **QuizForm**: genera dinamicamente input basandosi sul tipo di domanda (string, enum, boolean, number). Gestisci la navigazione "Avanti/Indietro", la memorizzazione delle risposte e la validazione per assicurare che ogni domanda sia compilata.
   - **LoadingGenerate**: simile a LoadingResearch ma con un focus sulla generazione del contenuto.
   - **ResultView**: mostra content_title, content_preview (in formato

markdown o testuale), metriche (parole, costo, tempo, agenti) e stato di consegna della mail. Fornisci pulsanti per scaricare il testo in formato .md e per ricominciare una nuova sessione.

5. **Transizioni di stato**
   - Usa un step interno nel componente OnboardingFlow per passare tra form, research, quiz, generate e result.
   - Aggiorna lo step automaticamente in base allo stato della sessione restituito dal backend: quando lo stato diventa awaiting_user mostra il quiz, quando è done mostra il risultato, e se è failed comunica l'errore all'utente.
6. **Gestione errori e UX**
   - Mostra banner o toast per errori di rete o validazione provenienti dall'API.
   - Per gli enum, imponi che il valore scelto corrisponda esattamente alle opzioni fornite (attenzione a maiuscole/minuscole).
   - Interrompi il polling e fornisci un messaggio adeguato quando la sessione non viene trovata (es. dopo 24 ore) o quando l'API restituisce un errore.
7. **Testing**
   - Verifica il flusso con cURL o Postman usando gli esempi forniti nella guida.
   - Implementa test manuali del frontend simulando diverse casistiche: input errati, sessioni fallite, sessioni con più domande, enumerazioni errate, ecc.
   - Se usi Next.js, puoi creare pagine con getServerSideProps per pre-compilare l'URL dell'API.
8. **Deployment**
   - Compila la versione di produzione (npm run build) e assicurati che l'host dell'API sia configurato tramite variabili d'ambiente.
   - Se deployi su Vercel o Netlify, imposta NEXT_PUBLIC_ONBOARDING_API_URL nei settings del progetto e assicurati che il backend supporti i domini di produzione nella configurazione CORS.
   - Rivedi la configurazione del backend per limitare allow_origins ai tuoi domini e assicurati che i microservizi (CGS, Perplexity, Gemini, Supabase) siano raggiungibili e configurati correttamente.

🔚 Conclusione

Il backend di onboarding del progetto CGS_2 fornisce un'API strutturata e tipizzata con FastAPIraw.githubusercontent.com, capace di orchestrare la ricerca sul brand, la sintesi di uno snapshot, la raccolta di risposte alle domande di chiarimento e l'esecuzione del workflow di generazione.
Per costruire il frontend:

1. Scegli e configura un framework (React/Next.js con TypeScript).
2. Implementa un client API e un hook di polling robusto.
3. Suddividi l'interfaccia in step chiari: form, research, quiz, generazione,

risultato.
4. Segui le regole di validazione e gestisci correttamente gli stati asincroni per offrire un'esperienza fluida all'utente.

Con questi elementi potrai trasformare la guida fornita in una soluzione completa e funzionante per l'onboarding della piattaforma CGS.