📄 IronMath – Documento Tecnico per Collaborazione Full-Stack & AI

0. Contesto e obiettivo del documento

Questo documento presenta in modo sintetico ma completo:

- la **vision** di IronMath
- il **perimetro della beta**
- l'**architettura tecnica** prevista (front, back, DB, LLM)
- la **divisione dei ruoli** tra noi founder e un possibile collaboratore senior full-stack/AI
- le **aspettative** rispetto a una consulenza iniziale e a una successiva collaborazione continuativa.

L'obiettivo è permettere al programmatore di:

1. capire con precisione **che cosa vogliamo costruire**
2. valutare la **fattibilità tecnica** dell'architettura
3. aiutarci a definire e implementare la parte critica **backend + LLM + DB + cloud**
4. lasciare a noi founder il massimo controllo possibile su **UX, contenuti, front-end e "mente didattica" dell'LLM**.

---

1. Visione generale di IronMath

IronMath è una webapp di supporto allo studio della matematica per studenti 11–18 anni (medie + liceo), con particolare attenzione a:

- studenti in difficoltà, anche con DSA
- studenti bravi che vogliono consolidare/accelerare

- genitori e professori che vogliono **visibilità sui progressi**

La vision:

- non usare l'AI come "solver di compiti", ma come **tutor socratico** che fa domande, guida, corregge il ragionamento, blocca le scorciatoie;
- integrare una **diagnostica forte** basata su prerequisiti:
  se non hai basi solide, IronMath ti blocca gli argomenti avanzati e ti rimanda indietro;
- coprire solo **matematica** nella prima fase, ma con un impianto serio, scalabile e adattato anche a studenti con DSA;
- diventare uno **standard informale** per professori e studenti ("Ragazzi, oggi 2h su IronMath e poi liberi").

La concorrenza diretta in Italia (youmath, chimicaonline, ecc.) è:

- statica, poco UX-oriented, non personalizzata
- non integrata con LLM, né con una vera diagnostica adattiva.

IronMath vuole essere il contrario: **dinamica, guidata, "umana", con AI al servizio della didattica**.

---

2. Scope della Beta

La **beta** deve dimostrare:

1. che la piattaforma può:
    - profilare lo studente
    - diagnosticare lacune
    - bloccare / sbloccare argomenti sulla base dei prerequisiti
    - guidare esercizi step-by-step
    - generare un report comprensibile per studente/genitore
2. che l'integrazione LLM:

    - è controllata
    - non “sputa la soluzione”
    - produce feedback coerente con il tono educativo e con lo stato dello studente.

2.1 Funzionalità minime della beta

- Registrazione/login con profilo studente:
    - medie / liceo
    - anno
    - indirizzo (scientifico / non scientifico)
    - obiettivo: sopravvivere / consolidare / eccellere
- Onboarding con breve **questionario diagnostico di base**
- Dashboard:
    - **Medie**: lezioni, formulario, esercizi guidati, esercizi non guidati, verifiche
    - **Liceo**: matematica e fisica, con logica simile
- **Grafo di prerequisiti** (skill graph) per argomenti:
    - se non superi il test per un nodo, non accedi al successivo (hard-lock) con possibilità di sblocco in extremis…
- Modalità **“Esercizi guidati”**:
    - multi step
    - pulsante “Avanti” / “Hint”
    - interazione con LLM come tutor
- Modalità **“Report”**:
    - riepilogo sessione con punti forti/deboli, suggerimenti, eventuali alert per genitori/prof

Non servono ancora:

- pagamenti
- area professori strutturata
- gamification avanzata (però direi che area trofei popup in stile Steam si può già implementare su basi sia statiche che LLM su progressi.
- app mobile nativa (per ora PWA / responsive)

---

3. Architettura tecnica – panoramica

A livello alto immaginiamo:

- **Front-end SPA** (inizialmente vanilla JS modulare / poi facilmente migrabile verso React/Next)
- **Backend REST** (Node.js + Express o stack equivalente)
- **Database relazionale** (preferenza PostgreSQL; in alternativa Mongo se più comodo)
- **LLM layer** (OpenAI/altro provider) incapsulato in un modulo unico tipo `llmService`
- **Deployment cloud** (Render, Vercel, AWS, ecc.)
- **Logging & analytics** (minimo iniziale, ma predisposto per crescita)

---

4. Front-end: stato attuale e fase 1

Abbiamo già iniziato la **Fase 1**:

- Single Page Application con:
    - `index.html` unico
    - `style.css` con palette “Iron Man” (rosso/oro, sfondo scuro moderno)
    - `app.js` che:
        - mantiene uno **state** globale (profilo studente + view corrente)

        - gestisce un piccolo router interno (“landing”, “onboarding”, “dashboard-medie”, “dashboard-liceo”)
        - inietta dinamicamente il contenuto nel `<main id="app-root">`

- Struttura logica delle view:
    - Landing → descrizione, call to action “Inizia ora”
    - Onboarding → scelta medie/liceo, obiettivi, sensazione verso matematica
    - Dashboard Medie/Liceo → pulsanti per lezioni, formulario, esercizi, verifiche

L’obiettivo è:

- dare subito al dev un quadro chiaro del **flusso utente**
- poter cambiare in futuro l’implementazione (vanilla → React) senza cambiare la **logica di prodotto**.

Cosa continueremo a fare noi sul front-end

- UI/UX
- styling, componenti, layout, responsive
- flussi di navigazione
- elementi di accessibilità base (font, contrasto, struttura chiara per DSA)
- integrazione con le API che ci verranno messe a disposizione

---

5. Backend & API – ruolo del dev vs ruolo dei founder

5.1 Backend – responsabilità principali del dev

Ci aspettiamo dal dev:

- scelta e setup dello stack backend (presumibilmente Node.js + Express)
- definizione dell’architettura API (routing, controllers, servizi)

- gestione sicurezza di base:
    - autenticazione
    - sanitizzazione input
    - rate limiting (soprattutto lato LLM)
- error handling coerente
- integrazione con DB
- integrazione con LLM (`llmService`)
- predisposizione al deploy cloud (Docker/CI/CD se necessario)

5.2 Endpoint chiave che immaginiamo

- `POST /api/auth/register`
- `POST /api/auth/login`
- `GET /api/exercises/:id` (e/o per argomento/skill)
- `POST /api/exercise/attempt`
- `POST /api/tutor/hint` → chiama l'LLM tutor con un contesto
- `POST /api/report` → chiama l'LLM reporter con lo storico della sessione
- `GET /api/profile/me`
- `POST /api/profile/update`
- `GET /api/skills/graph` (per recuperare info sui prerequisiti)

5.3 Cosa possiamo gestire noi sulle API

Noi due founder possiamo occuparci di:

- **API semplici** lato logica:

    - endpoint che fanno basic CRUD (profili, configurazioni, salvataggio preferenze)
    - endpoint che leggono esercizi dal DB e li presentano al front
- wiring front-end ↔ API
- test delle API via Postman/Thunder Client
- gestione di piccole evoluzioni (aggiunta campi, piccoli cambi di payload)

Per **API semplici** intendiamo:

- niente logiche di sicurezza avanzata
- niente integrazione con LLM
- niente logiche transazionali complesse

---

6. Database – struttura, ruoli e domande specifiche al dev

Vogliamo un database che supporti:

- utenti/profili
- esercizi
- skill graph (prerequisiti)
- tentativi
- log delle interazioni LLM
- report

6.1 Tabelle/collezioni di base (proposta)

- `users`
- `profiles` (impostazioni studente, livello, obiettivo, DSA sì/no)

- `skills` (ogni nodo del grafo: es. “Equazioni primo grado”, “Vettori”, ecc.)
- `skill_links` (relazioni prerequisito → successore)
- `exercises` (metadati esercizi)
- `exercise_steps` (passi guidati di un esercizio)
- `student_attempts` (storico tentativi)
- `llm_logs` (input/output chiamate al modello)
- `reports` (report generati per studente/sessione)

6.2 Cosa chiediamo al dev sul DB

Vorremmo:

1. **Definizione dello schema** (PostgreSQL o altro DB consigliato) con:
    - tipi, chiavi primarie, relazioni, indici
    - suggerimenti su come modellare il grafo dei prerequisiti
2. Una spiegazione chiara di:
    - quali tabelle NON dovremmo toccare a cuor leggero (es. `llm_logs`, `users`)
    - quali tabelle possiamo popolare/manipolare noi (es. `exercises`, `skills`, `exercise_steps`)
3. Una pipeline di **migrazione** chiara:
    - dove si definiscono gli schema
    - come si applicano le migrazioni
    - come aggiungere nuovi campi in modo sicuro

6.3 Cosa possiamo fare noi sul DB

Grazie al background in logica e informatica, possiamo:

- popolare le tabelle di contenuto (esercizi, skills, grafo dei prerequisiti)

- fare query semplici per debugging (`SELECT`, `JOIN` base)
- scrivere script di seed
- proporre evoluzioni sul modello dati legate alla didattica

Non vogliamo intervenire in:

- tuning di performance
- gestione indici avanzati
- scelte di normalizzazione/denormalizzazione critiche
- configurazione infrastrutturale del DB in produzione

---

7. LLM – inizializzazione a cura del dev, “allenamento” continuo a cura dei founder

7.1 Cosa ci aspettiamo dal dev sul LLM

- scelta del provider/model OpenAI capendo bene se è il migliore
- implementazione di un modulo unico tipo `llmService` che offra:
    - `getTutorHint(context)`
    - `getLearningReport(context)`
- definizione chiara del **formato del `context`**:
    - chi è lo studente
    - quale esercizio
    - quali step già affrontati
    - quali errori ha fatto
    - quale è il livello/obiettivo

- gestione robusta di:
    - errori di rete / timeout
    - risposta vuota / incoerente
    - rate limit
    - logging delle richieste/risposte
- parametri di base:
    - modello
    - temperature
    - max tokens
    - eventuali sistemi di retry

7.2 Cosa vogliamo gestire noi sull'LLM

Una volta che il dev ha impostato **il "motore LLM"**, noi possiamo:

- scrivere e iterare i **prompt di sistema** ("sei un tutor di matematica per 11–18 anni…")
- definire esempi (few-shot) per:
    - hint "soft"
    - hint "più diretti"
    - feedback "stai andando fuori strada"
- progettare il **tono di voce**:
    - empatico ma fermo
    - adatto a chi ha ansia/disagio
    - adatto anche a DSA
- aggiornare i prompt e i contesti in modo iterativo, sulla base dei risultati reali
- sperimentare varianti di:

    - prompt per medie
    - prompt per liceo
    - prompt per “studente forte” vs “studente in difficoltà”

In futuro, quando avremo abbastanza dati, valuteremo anche:

- fine-tuning
- dataset più strutturati

Ma nella beta ci basta un **LLM ben integrato e fortemente guidato dai prompt**.

---

8. Divisione dei compiti – cosa conviene gestisca il dev, e cosa noi

8.1 Il dev dovrebbe principalmente gestire:

- architettura backend
- sicurezza di base (auth, sanitizzazione, rate limiting LLM)
- implementazione `llmService`
- modello dati principale
- setup e configurazione del DB in ambiente di sviluppo e produzione
- log centralizzati e strutturati
- deploy (Docker/CI/CD, hosting, variabili d’ambiente)

8.2 Noi due founder vogliamo gestire:

- concept, UX, struttura dei flussi
- front-end (HTML, CSS, JS, componenti, router, layout)
- parte di API semplici (CRUD, end-point di lettura/scrittura non critici)

- contenuti matematici (teoria, esercizi, testi)
- skill graph/prerequisiti
- design del comportamento del tutor (prompt, esempi, policy didattica)
- test intensivi sull'esperienza studente

8.3 Obiettivo comune MAIN GOAL SEMANTICO

- il dev costruisce la **spina dorsale tecnica** stabile;
- noi costruiamo la **mente didattica** e l'interfaccia con lo studente.

---

9. Cosa chiediamo in una prima consulenza

Nella prima fase di collaborazione vorremmo:

1. **Revisione critica** di questa architettura proposta e del progetto globale
    - eventuali cambi di stack suggeriti
    - rischi e punti deboli
    - stime di complessità
2. **Proposta di schema DB** di prima versione + istruzioni operative su come popolarlo
3. **Specifiche di `llmService`**:
    - firma delle funzioni
    - formati dei parametri
    - esempi di payload di request/response
4. **Definizione chiara di chi fa cosa** sulle API:
    - lista di endpoint che conviene scriva lui
    - lista di endpoint che possiamo implementare noi

5. **Suggerimento infrastruttura**:
    - ambiente di sviluppo + editor e previewer della scrittura dei contenuti lezioni/esercizi/verifiche…
    - ambiente di staging
    - prima idea di deploy produttivo (anche solo “semi-artigianale”)
6. Un piano realistico per arrivare a una **beta funzionante in 60–90 giorni**, con:
    - milestone tecniche
    - milestone di contenuto
    - responsabilità condivise

---

10. Conclusione

Siamo due studenti di Cybersecurity con:

- buona base in HTML/CSS/JS
- infarinatura sui framework front-end
- competenze logiche e matematiche (inclusa algebra lineare)
- forte motivazione a **capire fino in fondo** l’infrastruttura, non solo a usarla

Vogliamo:

- imparare dal lavoro con un professionista
- mantenere il controllo su **brand, UX, contenuti, comportamento didattico dell’LLM**
- affidarci a un esperto per le parti in cui l’esperienza decennale fa davvero la differenza (backend, AI integration, cloud, sicurezza)

L’obiettivo è costruire insieme **una beta solida, scalabile e pulita**, che dimostri il valore di IronMath e che possa evolvere rapidamente nelle fasi successive.