Specifiche progetto:

Roadmap d'implementazione (visione d'insieme)

1. **Setup progetto** → struttura repo, ambienti, config.
2. **Ingestione dati** → scarico corpora, pulizia, deduplica, allineamenti Simple/Standard.
3. **Annotazione testi** → calcolo leggibilità (es. Gulpease per IT, Flesch/FKGL per EN), lunghezze, sintassi.
4. **Rappresentazione semantica** → embedding di documenti e di interessi utente.
5. **Catalogo** → storage (CSV/Parquet o DB), indici, API interne.
6. **Modello utente** → cold-start, profilo tematico, target di leggibilità, storico feedback.
7. **Motore di raccomandazione (baseline)** → scoring (tema + leggibilità), ranking, filtri.
8. **Feedback** → logging di tempo, completamento, abbandono, rating opzionale.
9. **Adattamento** → aggiornamento del target di leggibilità e dei pesi in base al feedback (progressione).
10. **Valutazione** → offline (NDCG@k, RMSE target-dev), online (CTR, completion, dwell).
11. **Messa in produzione** → API service, caching, monitoring, versionamento modelli/dati, privacy.

---

0) Prerequisiti e obiettivi chiari

- **Obiettivo tecnico misurabile**: ridurre la **deviazione media** tra la leggibilità dei testi letti e il **target utente** sotto una soglia (es. ≤ 5 punti Gulpease) **senza** ridurre l'engagement (completion ≥ 70%).
- **Obiettivo didattico**: far crescere il **target di leggibilità** medio dell'utente di X punti in Y settimane **senza incremento di abbandoni**.

---

1) Setup progetto

1.1 Struttura repository

```
readability-navigator/
├─ data/
│  ├─ raw/        # dump originali
│  ├─ interim/    # dopo pulizia/parsing
│  └─ processed/  # con feature finali
├─ conf/
│  ├─ project.yaml  # path, seed, k, soglie
│  └─ model.yaml    # modelli embedding, pesi η/ζ
```

```
├─ notebooks/       # analisi esplorative (facoltativo)
├─ src/
│  ├─ ingest/      # scarico + parsing corpora
│  ├─ features/    # leggibilità, embedding, topic
│  ├─ catalog/     # storage + indici + API interne
│  ├─ users/       # profilo, cold-start, update
│  ├─ recommender/ # scoring, ranking, bandit
│  ├─ feedback/    # logging, normalizzazione segnali
│  └─ eval/        # metriche offline/online
├─ tests/
└─ README.md
```

1.2 Ambienti

- Ambiente **CPU** per preprocessing e metriche; **GPU** solo per embedding se necessario.
- Fissa un **random seed** globale (riproducibilità).
- File .env per credenziali/API (se servono).

1.3 Configurazioni chiave (esempi)

- k (num raccomandazioni) = 3–5.
- Pesi di scoring: **η** (tema) = 0,6; **ζ** (carico) = 0,4 (parti così, poi farai tuning).
- Soglie: completion buono ≥ 0,9; abbandono alto ≥ 0,5; dwell time "troppo lungo" = > 95° percentile per quel livello.

---

2) Ingestione dati

2.1 Scelta corpora

- **Simple English Wikipedia** e **Standard Wikipedia**: consente pairing di testi su stesso tema con diversa difficoltà.
- **ASSET / OneStopEnglish** (frasi/articoli multilevel) per convalidare la pipeline di difficoltà.
- (Italiano) **Wikipedia IT** per testare Gulpease — anche senza parallelismi.

2.2 Scarico

- Per Wikipedia: usa le API o dump; limita a **categorie tematiche** utili alla scuola (scienze, storia, geografia).
- Mantieni **ID stabile** per titolo+lingua+rev.

2.3 Parsing & pulizia

- Rimuovi markup (HTML/Wiki), box laterali, elenchi tecnici troppo rumorosi.
- Normalizza: lowercasing (ma preserva acronimi se misuri leggibilità), rimuovi note [1], [2], ecc.

2.4 Deduplica & allineamento

- Deduplica per **Jaccard** su n-grammi o **cosine** su embedding grezzi.
- Allinea Simple/Standard su titolo; conserva mapping doc_id_standard → doc_id_simple.

Output atteso in data/interim/: JSON/CSV con doc_id, lang, title, body_clean.

---

3) Annotazione testi (feature linguistiche)

3.1 Metriche di leggibilità

- **Italiano**: **Gulpease**. Formula (usando parole, frasi, lettere):

$$G = 89 + \frac{300 \cdot S - 10 \cdot L}{P}$$

dove S= frasi, L= lettere, P= parole.
(Più alto → più facile; 80 facile, ~60 medio, <40 difficile.)

- **Inglese**: Flesch Reading Ease e/o FKGL, Gunning Fog.
- **Aggiuntive**: lunghezza media frasi, varianza lunghezze, proporzione subordinate, densità lessicale (tipo/token), % parole lunghe.

3.2 Normalizzazione

- Porta tutte le metriche su **scale comparabili** (z-score per corpus/lingua).
- Crea una **leggibilità aggregata** (es. media pesata di 2–3 indici; pesi determinati su set di validazione).

3.3 Controlli qualità

- Escludi documenti con < 120 parole (instabili per la metrica).
- Tagga documenti > 2000 parole come *lunghi* (feature binaria).

Output in data/processed/docs_features.parquet con:
[doc_id, title, lang, tokens, sentences, letters, gulpease/flesch, len_frasi, long_doc_flag, ...]

---

4) Rappresentazione semantica (tema/argomento)

4.1 Scelta embedding

- Document-level **sentence embedding** (es. modello multilingue se mischi IT/EN).
- Estrarre embedding di:
    - **Titolo** (forte segnale di tema)
    - **Primi N paragrafi** (riassunto semantico)

- **Intero testo** (se fattibile, oppure media di chunk).

4.2 Costruzione vettore finale

- Concatena o media pesata (es. 0,7 titolo+lead, 0,3 corpo).
- **L2-normalize** i vettori → cosine più stabile.

Output in data/processed/docs_embeddings.npy + mapping in .parquet.

---

5) Catalogo e storage

5.1 Schema (DB o files)

- **Docs**: doc_id (PK), lang, title, url, topic_vector, readability_vector, length_tokens, flags.
- **Pairs** (facoltativo): relazioni Simple↔Standard.
- Indici su lang, title, readability_bucket (buckets da 10 punti), e **ANN index** (es. FAISS) sul topic_vector per retrieval veloce.

5.2 API interne (servizi del tuo backend)

- get_docs_by_topic(query_vector, topN) → recupera candidati tematici.
- filter_by_readability(cands, target, tol) → filtra per fascia [target−tol, target+tol].
- rank_docs(cands, user_profile) → applica scoring finale (vedi §7).

---

6) Modello utente

6.1 Cold-start

- Mini-questionario: 3–5 argomenti preferiti (scegli da tassonomia semplice).
- **Seed target leggibilità**: breve test o mediana corpus "facile" (es. 60 Gulpease IT o 70 Flesch EN).
- **Embedding interessi**: media degli embedding dei titoli scelti (o di 2–3 testi seed).

6.2 Profilo dinamico (dopo le prime letture)

- target_readability_u (float)
- topic_vector_u (embedding medio pesato dalle letture completate, peso ∝ gradimento o dwell)
- Historico: reads_u = [{doc_id, start_ts, end_ts, completion, dwell, rating?}]

Persisti il profilo in DB o KV store con **versione**.

---

7) Motore di raccomandazione (baseline solida)

7.1 Retrieval candidati (tema)

- Calcola sim_tema = cosine(user.topic_vector, doc.topic_vector).

- Prendi **top M** candidati per tema (es. M=200).

7.2 Filtro per carico cognitivo

- Stima scostamento: dev = |readability(doc) − target_readability_u|.
- Filtra con una **tolleranza** iniziale (es. tol = 7 punti Gulpease/Flesch).
- Se candidati < K, allarga tol progressivamente (+3, +5).

7.3 Scoring

$$\text{score}(u,d) = \eta \cdot \text{sim\_tema}(u,d) \; - \; \zeta \cdot |\, \text{readability}(d) - \text{target}_u \,| \; + \; \lambda \cdot \text{novelty}(u,d)$$

- novelty(u,d): penalizza documenti troppo simili a quelli appena letti (es. −cosine con ultimi 3).
- Valori di partenza: η=0,6, ζ=0,4, λ=0,05.

7.4 Reranking (optional ma utile)

- **Diversificazione**: applica *MMR* per evitare 3 articoli quasi identici.
- **Lunghezza**: preferisci *medio-corti* quando il target è basso.

7.5 Esplorazione

- **ε-greedy** (ε=0,1): 10% dei casi inserisci 1 documento "di frontiera" (target+Δ piccolo) per testare capacità.

Output: lista top-K (K=3–5) con motivazioni ("coerente con *energia solare* e difficoltà vicina al tuo livello").

---

8) Feedback e tracciamento

8.1 Eventi minimi da loggare

- impression (quali doc sono stati mostrati),
- click/open (quale doc è stato aperto),
- **dwell time** (tempo effettivo sulla pagina),
- **completion rate** (proporzione di testo letta; se non hai scroll preciso, usa tempo ≥ soglia come proxy),
- rating opzionale (like/neutral/dislike).

8.2 Normalizzazione

- Rimuovi outlier (tab cambiate in background, ecc.).
- Storico aggregato per **sessione** e **utente**.

---

9) Adattamento (progressione del carico)

9.1 Regole semplici (robuste)

- Se completion ≥ 0,9 **e** dwell ≤ t95(level) → **alza** target_readability_u += Δ (Δ=2).
- Se abandon ≥ 0,5 **o** dwell > t99(level) → **abbassa** target_readability_u -= Δ (Δ=2).
- Clamp del target in [min_level, max_level] per lingua/corpus.

9.2 Aggiornamento tema

- Aggiorna topic_vector_u con media esponenziale pesata dalle letture **completate** (peso ∝ gradimento).

9.3 Tuning dei pesi η/ζ

- Se l'utente "salta" spesso → aumenta ζ (dai più peso alla differenza di difficoltà).
- Se si annoia (completion alto ma rating basso) → aumenta η (più tema).

---

10) Valutazione

10.1 Offline (prima di esporre agli utenti)

- **Splitting**: per utente in **leave-last-N** (es. lascia l'ultima lettura come test).
- **Label implicite**: positive = completate, negative = abbandonate/mai aperte.
- **Metriche**:
  - **NDCG@k**, **Recall@k** (classiche RS)
  - **Target-Deviation@k**: media | $read(doc_k) - target_u$ |(più basso = meglio)
  - **Calibration**: differenza media tra distribuzione difficoltà raccomandata e target desiderato
  - **Coverage**, **Novelty** (per non overfit su pochi doc)

10.2 Online (pilota)

- **CTR** (impression→open), **Completion**, **Dwell**
- **Lift** vs baseline (lista casuale filtrata per tema)
- Test **A/B**: baseline vs progressione attiva (update target).

---

11) Interfaccia e accessibilità (minimo indispensabile)

- Viewer testo con: **font dislessico**, spaziatura 1.5–2.0, larghezza riga contenuta, tema ad alto contrasto, TTS on-demand.
- Pannello "Perché questo testo?": mostra **tema corrispondente** e **livello adatto al tuo profilo** (spiegabilità).
- Controlli di **consenso** (profilazione, log).

---

12) API/Servizi (contratti chiari)

- POST /profile/init → crea profilo con interessi e seed target
- GET /recommend?user_id=U&topic=T&k=K → restituisce top-K con motivazioni
- POST /feedback → invia eventi (impression, open, dwell, completion, rating)
- POST /profile/update → endpoint interno (batch) che aggiorna target_readability e topic_vector

---

13) Scalabilità e indici

- Pre-calcola embedding e indici ANN per il retrieval tematico.
- Caching per: (utente, topic) → top-200 candidati 10–30 min.
- Batch notturno per ricalcolo metriche di leggibilità se aggiorni il catalogo.

---

14) Privacy & sicurezza

- Eventi e profili **pseudonimizzati** (user_id random).
- Nessun dato sensibile in chiaro.
- **Retention** breve dei raw logs, solo metriche aggregate per training.
- Pannello "scarica/elimina i tuoi dati".

---

15) Testing & qualità

- **Unit test**: parsing, calcolo metriche, scoring monotonicità (se target aumenta, doc troppo complicati devono scendere).
- **Data tests**: nessun doc con words < 120, nessun embedding NaN, range leggibilità valido.
- **E2E**: flusso "richiedi topic → vedi consigli → leggi → feedback → vedi progressione".

---

16) Estensioni (facoltative ma forti)

- **Sequence-aware RS**: modello che tiene memoria della *traiettoria* (es. GRU4Rec) per gestire la progressione.
- **Contextual bandit**: regola in tempo reale η/ζ in base al reward implicito.
- **Cross-lingua**: se IT è scarso, proponi testi EN *facili* con TTS, controllando ancora il carico.
- **Spiegazioni**: estrai 1–2 frasi "ancora complesse" per consigliare glossario (non riscrivere, ma *accompagnare*).

---

17) Parametri iniziali (che poi tunerai)

- k=3 (raccomandazioni mostrate)
- M=200 (candidati tematici)

- tol=7 (fascia iniziale intorno al target)
- η=0,6, ζ=0,4, λ=0,05
- Δ=2 (step di aggiornamento target)
- ε-greedy ε=0,1

---

18) Checklist per partire a scrivere il codice (in ordine)

1. **Scarica** 500–2.000 articoli da Simple/Standard (1–3 domini tematici).
2. **Pulisci e parse** in data/interim/.
3. **Calcola** metriche di leggibilità + feature base → data/processed/docs_features.parquet.
4. **Estrai** embedding semantici → data/processed/docs_embeddings.npy.
5. **Costruisci** catalogo + indici (in memoria o DB leggero).
6. **Implementa** profilo utente (cold-start + storage).
7. **Implementa** retrieval → filtro → scoring → reranking → top-K.
8. **Aggiungi** logging feedback e **update** target.
9. **Valuta** offline con split leave-last-N; calcola NDCG@k e Target-Deviation@k.
10. **Itera** su η/ζ/tol/Δ finché non hai: *calibration buona* + *completion* ≥ *70%* nei test pilota.