DOCUMENTO TECNICO DI PROGETTO

1. Analisi della problematica

L'obiettivo del progetto è la realizzazione di un sistema software per la raccolta, la normalizzazione e l'analisi dei dati relativi al settore economico della pesca in Italia, con particolare attenzione alla presentazione dei risultati in modo accessibile tramite API e viste grafiche/tabellari. Il sistema deve essere in grado di: - Importare dati pubblici da file CSV eterogenei. - Eseguire pulizia, normalizzazione e interpolazione sui dati. - Aggregare i dati su base geografica (aree italiane) e temporale (anno). - Esporre i dati e le serie calcolate tramite API REST. - Consentire la visualizzazione dei risultati in formato grafico.

1. Scelte progettuali

Linguaggi e tecnologie - Python: scelto per la sua semplicità nella gestione dei dati (libreria pandas) e la disponibilità di framework web moderni (FastAPI). - FastAPI: framework performante, moderno e facilmente integrabile con OpenAPI/Swagger, ideale per la creazione di API REST. - SQLite: database relazionale leggero, ideale per prototipazione, demo e piccoli/medi volumi di dati. Permette di avere il database "self-contained" e facilmente distribuibile. - Pandas: per tutte le fasi di ETL (estrazione, trasformazione, caricamento) e le operazioni di analisi/statistica. - Chart.js / Matplotlib: scelti per la visualizzazione grafica sia lato frontend (Chart.js) sia per eventuali grafici statici (Matplotlib). - Jupyter Notebook: utilizzato per esplorazione e prototipazione rapida durante le prime fasi di analisi.

Struttura del progetto - Suddivisione in cartelle: - services/: servizi dedicati a ciascuna serie calcolata o funzione aggregata. - utils/: classi e funzioni di utilità per ETL (es. CsvLoader, CsvRenamer, CsvNumericCaster, CsvExporter, ecc.). - data/: file del database SQLite. - statics/: file dati (CSV originali e output). - notebooks/: prototipazione e analisi interattiva. - Ogni componente ETL rispetta il principio della single responsibility.

Gestione dati e normalizzazione - I dati sono stati importati da CSV pubblici. - Le colonne sono state rinominate per coerenza e facilità di utilizzo. - Le colonne numeriche con virgola come separatore decimale sono state normalizzate (convertite a float). - I dati mancanti sono stati trattati con interpolazione lineare, solo quando il contesto (temporale/regione) lo permetteva, come richiesto. - Le regioni sono state collegate a un'area geografica tramite una tabella di mapping (regioni), secondo la suddivisione Nord-ovest, Nord-est, Centro, Sud, Isole.

Persistenza e modello dati - I dati normalizzati vengono salvati su SQLite in tre tabelle principali: - andamento - importanza - produttivita - È stata aggiunta la tabella regioni per il mapping geografico (id, nome, area). - Le serie calcolate vengono generate "on demand" via API e possono essere anche salvate come tabelle aggiuntive.

Calcoli e API - Tutte le serie calcolate sono implementate come servizi separati che aggregano i dati in base all'anno e all'area geografica. - Le API espongono: - Le tabelle di base (con filtro per intervallo anni). - Le serie calcolate (con filtro per intervallo anni). - L'esposizione API è conforme allo standard REST (JSON). - È stato aggiunto il middleware CORS per abilitare l'accesso frontend da altre porte/domini.

Visualizzazione - Il frontend utilizza Chart.js per consumare le API e mostrare i dati come grafico interattivo. - È previsto un endpoint di esempio anche per la generazione di grafici matplotlib direttamente come immagine (PNG).

1. Scelte non ovvie e assunzioni

2. Non è stato controllato l'univocità dei dati per ogni record anno-regione, perché i dati pubblici erano già organizzati in modo tabellare e aggregato per queste chiavi.

3. Non sono stati implementati controlli di coerenza tra percentuali e valori assoluti (mancando le quantità di riferimento nei dataset originali).
4. Interpolazione lineare adottata per dati temporali mancanti, come suggerito nel testo del problema.
5. Scelte di nomenclatura: si è adottato l'underscore ("snake_case") per i nomi delle colonne e delle variabili per coerenza con gli standard Python/Pandas.
6. Database scelto: SQLite è stato preferito per semplicità, leggerezza e facilità di integrazione con pandas e per evitare la necessità di un server dedicato.
7. Gestione concorrenza DB: ogni accesso a SQLite viene fatto con connessioni temporanee per evitare problemi di threading, secondo best practice FastAPI.

8. Non è stata realizzata una gestione degli utenti/autenticazione perché non richiesta dal progetto.

9. Fonti, ispirazioni e riferimenti

10. Documentazione ufficiale:

11. FastAPI: https://fastapi.tiangolo.com/
12. pandas: https://pandas.pydata.org/
13. Chart.js: https://www.chartjs.org/
14. Algoritmi di interpolazione e data cleaning:
15. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html

16. Progetti simili open-source su GitHub per struttura servizi e ETL in Python.

17. Conclusioni

Il progetto realizza una pipeline completa di raccolta, normalizzazione, analisi e visualizzazione dati, facilmente espandibile e adattabile ad altri dataset simili. La struttura a servizi permette una manutenzione semplificata e la possibilità di aggiungere nuove serie calcolate e API con poco sforzo. La scelta di Python/FastAPI e SQLite garantisce un'ottima accessibilità sia per chi sviluppa sia per chi utilizza i dati.