UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

DOCUMENTAZIONE TECNICA DEL SOFTWARE:

NoSchoolQL

A CURA DI:

Antonio Di Giorgio

Matricola: 0522502003

Anno Accademico 2024-2025

Indice

Elenco delle Tabelle **iv**

1 Introduzione **1**

1.1 Contesto didattico e accademico 1

1.2 Motivazioni del progetto 2

1.3 Scopo del documento 2

1.4 Struttura del documento 3

2 Definizioni e acronimi **4**

2.1 Glossario dei termini principali 4

2.2 Acronimi utilizzati 6

3 Contesto e obiettivi del progetto **7**

3.1 Requisiti didattici (proprietà BASE, CAP) 7

3.2 Vincoli funzionali e non funzionali 8

3.3 Ambito di applicazione e limitazioni 9

4 Descrizione del Miniworld **10**

4.1 Selezione degli istituti secondari di secondo grado 10

4.2 Dataset MIM originali elaborati 11

4.3 Entità semantiche del Miniworld 12

4.4 Relazioni implicite 12

5 Gestione dei dataset **13**

5.1 Orchestrazione generale con main.py 13

5.2 Pulizia e normalizzazione (pulizia_mim.py) 14

5.3 Calcolo preliminare delle statistiche (calcolo_statistiche.py) 14

5.4 Generazione dei dati simulati (genera_dati_simulati.py) 15

5.4.1 Caricamento dei dati di base 15

5.4.2 Generazione delle classi 16

5.4.3 Simulazione degli studenti e calcolo ESCS 16

5.4.4 Assegnazione delle materie 17

5.4.5 Generazione dei docenti 17

5.4.6 Calcolo dei voti 18

5.4.7 Esportazione e verifica 20

6 Architettura del sistema **21**

6.1 Vista high-level e flussi informativi 21

6.2 Backend (Node.js + Express) 22

6.3 Frontend (React) 22

6.4 Database NoSQL (MongoDB) 23

6.5 Deployment e ambiente operativo 24

7 Use Case e Requisiti Utente **25**

7.1 Attori principali 25

7.2 Use Case Studente 26

7.2.1 UC1 – Login Studente 26

7.2.2 UC2 – Visualizzazione Dashboard Studente 26

7.2.3 UC3 – Dettaglio Voti per Materia 26

7.3 Use Case Docente 27

7.3.1 UC4 – Login Docente 27

7.3.2 UC5 – Selezione Classe e Materia 27

7.3.3 UC6 – Inserimento Voti di Classe 27

7.3.4 UC7 – Modifica e Cancellazione di un Voto 28

7.4 Requisiti Funzionali 28

7.5 Requisiti Non Funzionali 28

8 Soluzione proposta e tecnologie **29**

8.1 Scelte architetturali principali 29

8.2 Tecnologie e librerie utilizzate 30

8.3 Integrazione di Faker e sistema di charting 31

8.4 Aggiornamenti on-demand dei dati 32

9 Metodologia di sviluppo **33**

9.1 Analisi e raccolta requisiti 33

9.2 Design tecnico 34

9.3 Implementazione 34

9.3.1 Pipeline ETL 34

9.3.2 Applicazione full-stack 35

9.4 Verifica e testing 35

9.5 Refactoring e mantenimento 36

10 Best Practice e Sicurezza **37**

10.1 Principi di codifica e manutenzione 37

10.2 Gestione di configurazioni e credenziali 38

10.3 Sicurezza delle API e convalida degli input 38

10.4 Protezione dei dati e conformità 39

10.5 Logging e auditing 39

10.6 Hardening dell'ambiente di esecuzione 39

11 Conclusioni **41**

11.1 Sintesi dei risultati 41

11.2 Limiti attuali 42

11.3 Conclusione 42

Bibliografia **44**

Elenco delle tabelle

2.1 Glossario dei termini principali 6

CAPITOLO 1

Introduzione

1.1 Contesto didattico e accademico

Il progetto NoSchoolQL_MIM nasce come esercitazione finale per l'insegnamento di Basi di Dati II presso l'Università di Salerno, corso di laurea magistrale in Sicurezza Informatica. Dopo aver acquisito i fondamenti teorici su modelli relazionali e NoSQL, durante il corso abbiamo appreso le proprietà BASE (Basically Available, Soft state, Eventual consistency) e il teorema CAP (Consistency, Availability, Partition tolerance). Attraverso NoSchoolQL_MIM, si è inteso sperimentare concretamente:

- **Modellazione documentale**: organizzare collezioni MongoDB in modo semantico, senza ricorrere a join complessi.
- **Pipeline dati**: processare dataset reali MIM con script Python per pulizia, normalizzazione e simulazione.
- **Architettura full-stack**: integrare un backend RESTful in Node.js/Express con un frontend React a pagina singola.

Questa esperienza supporta la comprensione delle criticità di consistenza e scalabilità in scenari reali, favorendo la padronanza di strumenti come MongoDB Compass, VSCode e Git/GitHub.

1.2 Motivazioni del progetto

Nel settore EdTech, la gestione dinamica dei registri elettronici richiede soluzioni flessibili e scalabili, in grado di:

1. Assorbire volumi elevati di dati eterogenei (anagrafica, voti, statistiche).
2. Offrire un'interfaccia reattiva e protetta per utenti con ruoli diversi (docenti, studenti, ospiti).
3. Garantire resilienza e accessibilità anche in presenza di partizioni di rete o doppie sessioni.

A partire dai CSV MIM, è stata costruita una pipeline modulare:

```
pulizia_mim.py -> calcolo_statistiche.py ->
    -> genera_dati_simulati.py -> analisi_dataset.py
```

ciascuno con responsabilità chiare e testabili. La componente applicativa è stata progettata per esporre API REST coerenti con le collezioni MongoDB generate, includendo:

- Endpoint CRUD per studenti, docenti, classi e voti.
- Autenticazione JWT e gestione dei permessi tramite middleware.

Tale struttura facilita l'estensibilità futura (nuove feature, integrazione mobile, microservizi).

1.3 Scopo del documento

La documentazione descrive architettura, componenti e flussi dell'applicazione, offrendo supporto tecnico a sviluppatori e manutentori, e garantendo la riproducibilità del processo di gestione dei dataset tramite descrizioni dettagliate degli script Python. Mira, inoltre, ad assicurare la tracciabilità tra requisiti, use case e implementazione. È rivolta a docenti, sviluppatori, DevOps e stakeholder, fornendo sia una visione tecnica sia funzionale del sistema. Realizzata secondo le linee guida IEEE 26511[1], garantisce conformità, completezza e tracciabilità, grazie a riferimenti incrociati tra requisiti, codice e documentazione tecnica.

1.4 Struttura del documento

Il documento segue un percorso logico che rispecchia il ciclo di vita del progetto:

1. **Definizioni e acronimi** – glossario termini principali e acronimi utilizzati.
2. **Contesto e obiettivi** – descrive vincoli teorici e limiti applicativi.
3. **Descrizione del Miniworld** – illustra le entità scolastiche e i dataset di partenza.
4. **Gestione dei dataset** – dettaglia gli script Python, mostrati con frammenti di codice e diagrammi di flusso.
5. **Architettura del sistema** – offre diagrammi componenti e spiegazioni di ogni modulo software.
6. **Use case e requisiti** – casi d'uso concreti per docente e studente, requisiti di sicurezza e performance.
7. **Soluzione proposta e tecnologie** – motivazioni per le scelte tecnologiche e librerie adottate.
8. **Metodologia sviluppo** – analisi e raccolta requisiti, design e implementazione, code review e refactoring.
9. **Best practice e sicurezza** – standard di codifica, sanitizzazione input e policy CORS.
10. **Conclusioni** – valutazioni finali del progetto.

Alla fine di ogni sezione troverai esempi tratti dal codice sorgente, diagrammi sintetici e tabelle riassuntive per facilitare la comprensione.

CAPITOLO 2

Definizioni e acronimi

Questa sezione fornisce chiarimenti sui termini tecnici e sugli acronimi utilizzati in tutto il documento, in modo da garantire una comprensione univoca dei concetti.

2.1 Glossario dei termini principali

TERMINE	**DEFINIZIONE**
NOSQL	Famiglia di DBMS non relazionali in cui i dati sono organizzati in strutture flessibili (es. documenti JSON), senza schema rigido delle tabelle relazionali.
COLLECTION	In MongoDB, insieme di documenti omogenei; l'equivalente di una "tabella" in un DB relazionale, ma senza vincoli di schema fisso.
DOCUMENT	Un record JSON-like memorizzato in una collection, composto da coppie chiave–valore e potenzialmente nidificato.
BASE	Principio di consistenza "soft" nei sistemi NoSQL: Basically Available, Soft state, Eventual consistency.

TERMINE	DEFINIZIONE
CAP	Teorema che afferma che un sistema distribuito può garantire al massimo due tra: Consistency, Availability e Partition tolerance.
PIPELINE	Sequenza di operazioni (lettura, trasformazione, scrittura) applicate ai dati grezzi per ottenere dataset puliti e normalizzati.
NORMALIZZAZIONE	Processo di standardizzazione dei valori (stringhe, formati data, nomi) per eliminare ambiguità e duplicati.
SIMULAZIONE DATI	Generazione di dati sintetici basati su distribuzioni e parametri socio-demografici per completare o estendere dataset reali.
RESTFUL API	Interfaccia di programmazione che espone risorse via HTTP, seguendo i principi REST (stateless, URL resource-oriented, uso di metodi standard).
CRUD	Operazioni fondamentali su dati persistenti: Create, Read, Update, Delete.
JWT	JSON Web Token; standard aperto per la trasmissione sicura di informazioni di autenticazione e autorizzazione tra client e server.
POLLING	Tecnica di interrogazione ripetuta (a intervalli regolari) di un endpoint per verificare aggiornamenti nello stato remoto.
CI/CD	Continuous Integration / Continuous Deployment: pratica di automazione del build, test e rilascio del software.
ETL	Extract-Transform-Load: paradigma per estrarre dati da sorgenti, trasformarli e caricarli in un sistema di destinazione.

TERMINE	DEFINIZIONE

Tabella 2.1: Glossario dei termini principali

2.2 Acronimi utilizzati

- **API**: Application Programming Interface
- **CSV**: Comma-Separated Values
- **DBMS**: DataBase Management System
- **EdTech**: Educational Technology
- **HTTP**: HyperText Transfer Protocol
- **JSON**: JavaScript Object Notation
- **JWT**: JSON Web Token
- **MIM**: Ministero dell'Istruzione e del Merito
- **NoSQL**: Not only SQL
- **PWA**: Progressive Web App
- **REST**: Representational State Transfer
- **UI**: User Interface
- **UX**: User Experience

CAPITOLO 3

Contesto e obiettivi del progetto

Questa sezione descrive il quadro teorico e pratico entro cui si colloca NoSchool-QL_MIM, illustrando i requisiti didattici derivanti dal corso, i vincoli imposti dal dominio applicativo e l'orizzonte delle funzionalità.

3.1 Requisiti didattici (proprietà BASE, CAP)

Il progetto è stato concepito per mettere in pratica due pilastri teorici fondamentali per i sistemi distribuiti NoSQL:

- **Proprietà BASE**
  - **Basically Available**: l'applicazione deve rimanere sempre raggiungibile, anche durante operazioni di manutenzione o carichi elevati, garantendo risposte rapide alle richieste REST.
  - **Soft State**: il sistema può trovarsi in uno stato transitorio (ad esempio, durante la sincronizzazione dei dati simulati), purché tenda rapidamente a uno stato coerente.
  - **Eventual Consistency**: dopo un aggiornamento (es. inserimento di un voto), ogni replica del database deve riflettere il cambiamento entro un intervallo di tempo prevedibile.

- **Teorema CAP**
  - **Consistency**: tutte le letture successive a una scrittura devono restituire il dato aggiornato. Nel nostro contesto, è accettabile una coerenza "eventuale" su statistiche non critiche, mentre sulle operazioni di voto l'endpoint `/api/registro/docente/voto` utilizza conferme sincrone.
  - **Availability**: ogni richiesta HTTP deve produrre una risposta (anche parziale), evitando timeout non gestiti a livello di client React.
  - **Partition tolerance**: il sistema deve continuare a funzionare anche in presenza di latenza di rete o partizioni tra server applicativo e istanze MongoDB, grazie alla configurazione di replica set di MongoDB e alla logica retry aggiunta nel middleware di connessione.

3.2 Vincoli funzionali e non funzionali

- **Vincoli funzionali**
  1. CRUD Studenti/Docenti/Classi/Voti: ogni risorsa espone operazioni Create, Read, Update, Delete via REST.
  2. Ruoli e autorizzazioni: solo i docenti possono inserire, modificare o cancellare voti; gli studenti hanno accesso in sola lettura al proprio storico.
  3. Filtri e statistiche: il frontend React offre filtri per materia e intervallo temporale, nonché grafici di distribuzione dei voti.
- **Vincoli non funzionali**
  1. Performance: tempo di risposta API inferiore a 2 sec su operazioni di lettura media (GET).
  2. Scalabilità: supporto a dataset fino a decine di migliaia di documenti (studenti e voti) senza degradare l'esperienza utente.
  3. Sicurezza: autenticazione JWT con scadenza configurabile, CORS limitato alle origini di frontend e header X-Auth token.
  4. Manutenibilità: codice organizzato in moduli (controllers, routes, script Python), con linter ESLint.

3.3 Ambito di applicazione e limitazioni

- **Ambito di applicazione:**
  - Destinato a istituti di istruzione secondaria con architetture full-stack leggere.
  - Si concentra sulla gestione di un singolo anno scolastico e su statistiche aggregate, senza funzionalità di pianificazione oraria o registro presenze.
- **Limitazioni:**
  - Temporale: il dataset simulato copre l'anno accademico 2023/2024 e non prevede aggiornamenti "live" oltre il termine del periodo di votazione.
  - Strutturale: non è prevista l'integrazione diretta con sistemi esterni di terze parti (es. registro ministeriale) né l'uso di microservizi multipli.

CAPITOLO 4

Descrizione del Miniworld

In questa sezione viene illustrata la prima fase di definizione del miniworld: l'estrazione degli istituti di Scuola Secondaria di Secondo Grado e l'esclusione dei file privi di tali dati.

4.1 Selezione degli istituti secondari di secondo grado

Nel file `pulizia_mim.py` si procede come segue:

1. **Lettura e preparazione di Stu_Corso_Classe_Genere.csv**

```
df_ordini = pd.read_csv(PATH_STU_CORSO_CLASSE, dtype=str)
df_ordini = clean_string_columns(df_ordini)
df_ordini = snake_case_columns(df_ordini)
```

2. **Estrazione dei codici delle scuole secondarie**

```
scuole_secondarie = set(
    df_ordini[df_ordini['ordinescuola']
        == 'SCUOLA SECONDARIA II GRADO']['codicescuola']
)
```

3. **Filtro dei dataset sui codici estratti** Tutti i DataFrame di anagrafica, studenti e indirizzi vengono filtrati mantenendo solo le righe il cui campo `codicescuola` è contenuto in `scuole_secondarie`.

Esclusione di Stu_Class_Num.csv

Il file `Stu_Class_Num.csv` non contiene righe con `ordinescuola = 'SCUOLA SECONDARIA II GRADO'` e pertanto viene completamente scartato, in quanto non rilevante per il dominio delle scuole secondarie di secondo grado.

4.2 Dataset MIM originali elaborati

Dopo il filtro, i file effettivamente candidati alle fasi successive sono:

- **AnagScuole.csv e AnagScuoleProvAutonome.csv**

  Concatenati e filtrati tramite:

```
anag1 = pd.read_csv(PATH_ANAG_SCUOLE, dtype=str)
anag2 = pd.read_csv(PATH_ANAG_SCUOLE_PA, dtype=str)
anag  = pd.concat([anag1, anag2], ignore_index=True)
anag  = clean_string_columns(anag)
anag  = snake_case_columns(anag)
anag  = anag[anag['codicescuola'].isin(scuole_secondarie)]
```

- **stu_cittadinanza_pulito.csv**

  Generato da `Stu_Cittad.csv`, contiene: `codicescuola`, `alunni`, `alunnicittadinanzaitaliana`, `alunnicittadinanzanonitaliana`.

- **stu_indirizzi_pulito.csv**

  Derivato da `Stu_Indirizzo.csv`, con colonne: `codicescuola`, `tipopercorso`, `indirizzo`, `alunnimaschi`, `alunnifemmine`.

- **stu_corso_classe_pulito.csv**

  Rappresenta proprio `Stu_Corso_Classe_Genere.csv` normalizzato e usato in questa fase per il solo filtro.

4.3 Entità semantiche del Miniworld

A partire dai file filtrati emergono tre unità concettuali fondamentali:

- **Scuola**: identificata da `codicescuola`, con attributi geografici e anagrafici.
- **Studenti (aggregati)**: rappresentati da conteggi di genere e cittadinanza a livello di scuola o indirizzo.
- **Percorso formativo**: definito da `tipopercorso` e `indirizzo`, utilizzato nella simulazione delle materie.

4.4 Relazioni implicite

Le associazioni sono basate esclusivamente sul campo `codicescuola`:

- Scuola $\rightarrow$ Studenti: collegamento tra anagrafica e dati demografici.
- Scuola $\rightarrow$ Percorsi: connessione tra istituto e indirizzi formativi.

Questo approccio, tipico dei database documentali NoSQL, evita join complessi tra tabelle, privilegiando letture dirette su collezioni filtrate per chiave primaria.

CAPITOLO 5

Gestione dei dataset

La pipeline di gestione dei dataset di NoSchoolQL_MIM è concepita come un processo ETL (Extract-Transform-Load) articolato in quattro fasi distinte, ciascuna implementata in uno script Python autonomo.
Il cuore di questa pipeline è `genera_dati_simulati.py`, che trasforma le tabelle preliminari in un set completo di collezioni documentali (classi, studenti, docenti, voti) corredate di distribuzioni realistiche e di fattori socio-demografici basati su studi ufficiali. Di seguito, ogni fase viene descritta nel dettaglio, mostrando i frammenti di codice reali e spiegando il razionale scientifico alla base di ogni scelta.

5.1 Orchestrazione generale con main.py

Il file `main.py` funge da coordinatore, eseguendo in sequenza i quattro script ETL e interrompendo la pipeline in caso di errore, per una più rapida individuazione del problema:

```
1 def esegui_script(nome_script, descrizione):
2     script_path = os.path.join(CURRENT_DIR, nome_script)
3     print(f"\nInizio: {descrizione} ({script_path})")
4     subprocess.run(['python', script_path], check=True)
5
```

```
fasi = [
    ('pulizia_mim.py', 'Pulizia dei file MIM'),
    ('calcolo_statistiche.py', 'Generazione statistiche per simulazione')
    ('genera_dati_simulati.py', 'Generazione dei dati simulati'),
    ('analisi_dataset.py', 'Analisi del dataset simulato')
]
for script, descrizione in fasi:
    esegui_script(script, descrizione)
```

Questa semplicità garantisce che ogni fase lavori sempre su dati coerenti e che eventuali errori non "inquinino" le fasi successive `genera_dati_simulati`.

5.2 Pulizia e normalizzazione (pulizia_mim.py)

La prima fase elimina ridondanze e uniforma i CSV grezzi del MIM. Dopo aver concatenato l'anagrafica delle scuole statali e delle province autonome, lo script applica la normalizzazione delle stringhe (rimozione di spazi e maiuscole) e converte i nomi delle colonne in formato snake_case, rendendo i filtri programmati più robusti `pulizia_mim`. Successivamente, viene ricavato il set `scuole_secondarie` filtrando il CSV `Stu_Corso_Classe_Genere.csv` sulle righe con `ordinescuola == 'SCUOLA SECONDARIA II GRADO'` e, su tale base, tutte le altre tabelle (anagrafica, cittadinanza, indirizzi) vengono ristrette a queste scuole, escludendo file privi di dati rilevanti come `Stu_Class_Num.csv`. Infine, in modalità ridotta, si esegue uno stratified sampling di 200 scuole per garantire tempi di esecuzione contenuti.

5.3 Calcolo preliminare delle statistiche (calcolo_statistiche.py)

Prima di simulare i dati, è indispensabile quantificare alcune medie "reali". Lo script `calcolo_statistiche.py` carica i CSV puliti e calcola per ciascuna scuola la percentuale di studenti maschi/femmine e di italiani/stranieri, salvando il risultato in `statistiche_base.csv`. Questi numeri forniscono i pesi di base per la distribuzione di genere e cittadinanza all'interno delle classi sintetiche, assicurando la coerenza con i dati MIM `genera_dati_simulati`.

5.4 Generazione dei dati simulati (genera_dati_simulati.py)

La fase di generazione dei dati simulati rappresenta il cuore della pipeline ETL: qui, partendo dalle tabelle preprocessate, si costruisce un mini-database NoSQL composto da quattro collezioni documentali – classi, studenti, docenti e voti – corredato di variabili socio-demografiche e didattiche plausibili. Il risultato finale è un insieme di CSV pronti all'importazione in MongoDB.

All'inizio del file, vengono definiti i percorsi di input/output, assicurandosi che la directory di destinazione esista, e si imposta un seed per i generatori casuali, in modo da garantire riproducibilità totale:

```
BASE_DIR   = os.path.dirname(os.path.abspath(__file__))
INPUT_DIR  = os.path.join(BASE_DIR, '../file/dataset_puliti')
OUTPUT_DIR = os.path.join(BASE_DIR, '../file/dataset_definitivi')
os.makedirs(OUTPUT_DIR, exist_ok=True)

if SEED is not None:
    random.seed(SEED)
    np.random.seed(SEED)

faker = Faker('it_IT')
if SEED is not None:
    Faker.seed(SEED)
```

In questo modo ogni esecuzione con `SEED=42` restituisce esattamente lo stesso dataset, cruciale per test e validazione `genera_dati_simulati`.

5.4.1 Caricamento dei dati di base

Vengono letti i tre CSV prodotti nelle fasi precedenti – anagrafica delle scuole, indirizzi con numeri di maschi/femmine e statistiche di base – e si normalizzano i campi numerici:

```
df_anag  = pd.read_csv(os.path.join(INPUT_DIR, 'anagrafica_scuole_pulita.
    csv'))
df_ind   = pd.read_csv(os.path.join(INPUT_DIR, 'stu_indirizzi_pulito.csv'
    ))
df_stats = pd.read_csv(os.path.join(INPUT_DIR, 'statistiche_base.csv'))
```

```
4
5 for col in ['alunnimaschi', 'alunnifemmine']:
6     df_ind[col] = df_ind[col].map(to_int_safe)
7 if 'totale' not in df_ind.columns:
8     df_ind['totale'] = df_ind['alunnimaschi'] + df_ind['alunnifemmine']
9 df_ind['indirizzo_norm'] = df_ind['indirizzo'].str.upper().str.strip()
```

Questi dataset costituiscono il punto di partenza per la creazione delle classi e degli studenti `genera_dati_simulati`.

5.4.2 Generazione delle classi

Per ogni riga di `stu_indirizzi_pulito.csv`, che racchiude il totale degli alunni per scuola, indirizzo e anno, si calcola il numero di classi semplicemente come il ceil del rapporto tra studenti e dimensione media:

```
1 num_classi = math.ceil(totale / MEDIA_ALUNNI_PER_CLASSE)
```

Si suddivide quindi il totale in gruppi equi, creando per ciascuno un record con:

- `id_classe`, un identificatore univoco generato incrementando un contatore per scuola,
- `nome_classe`, con anno + lettera sequenziale (A, B, ...),
- conteggi di maschi, femmine, italiani e stranieri,
- attributi geografici (provincia, area) estratti dal codice meccanografico `genera_dati_simulati`.

Questa prima collezione, esportata in `classi.csv`, rappresenta la struttura portante su cui si basa tutta la simulazione.

5.4.3 Simulazione degli studenti e calcolo ESCS

Con le classi definite, si passa a popolare la collezione studenti. Per ogni classe si ricavano due liste di etichette (['M', 'F', ...], ['ITA', 'UE', 'NON_UE', ...]) secondo le percentuali reali registrate nel file `statistiche_base.csv`. Queste liste vengono mescolate con `random.shuffle()` per eliminare pattern algoritmici.

Ogni studente riceve poi un nome e cognome: se italiano, li genera Faker; in caso di cittadinanza straniera, seleziona nomi e cognomi da insiemi predefiniti (es. ['Mohamed', ...], ['Singh', ...]) per maggiore realismo `genera_dati_simulati`.

L'aspetto distintivo è il calcolo del ESCS (Economic, Social and Cultural Status). La funzione `genera_escs_studente` parte da una normale standard e applica modificatori tratti da letteratura nazionale:

- Area geografica: Nord-Est +0.6, Nord-Ovest +0.4, Centro +0.2, Sud –0.4, Isole –0.5 [2, 3, 4],
- Tipo di scuola: liceo +0.3, istituto professionale –0.4 [3, 5],
- Cittadinanza: UE –0.2, NON_UE –0.6 [3, 4, 6] .

Il valore ottenuto viene clippato tra –2.86 e +1.78 e suddiviso in quartili con `calcola_escs_quartile`, restituendo un'attribuzione coerente con le soglie ufficiali. L'insieme di questi dati popolari compone `studenti.csv`, con ogni studente identificato da STU000001, STU000002 e così via, corredato di sesso, cittadinanza, ESCS e quartile.

5.4.4 Assegnazione delle materie

Definita la platea degli studenti, è cruciale specificare quali materie ogni classe insegni. La funzione `materie_per_classe` accede al dizionario `MATERIE_INDIRIZZO`, che mappa indirizzo e biennio/triennio a elenchi di materie ministeriali (es. Latino, Disegno, Fisica). Se l'indirizzo non è presente, si utilizza un fallback di materie comuni (Italiano, Matematica, ...) per garantire completezza `genera_dati_simulati`.

Questa mappatura produce un dizionario `materie_classe` che verrà usato nella generazione di docenti e voti.

5.4.5 Generazione dei docenti

Il "carico didattico" è determinato dividendo il totale delle cattedre (ogni classe × materia) per `MEDIA_CLASSI_PER_DOCENTE`. A ciascun docente, generato con nome e cognome casuali da Faker, si assegna un numero casuale di classi tra 3 e 6. Se qualche materia rimane senza insegnante, il codice crea docenti aggiuntivi fino

a copertura completa, garantendo che ogni classe-materia abbia almeno un docente `genera_dati_simulati`. Il risultato è il file `docenti.csv` e `assegnazioni_docenti.csv`.

5.4.6 Calcolo dei voti

La generazione del voto in `genera_dati_simulati.py` avviene all'interno della funzione `voto_generato(id_studente, id_classe, materia, tipologia)`, che combina in modo lineare diversi fattori per riprodurre fedelmente le dinamiche reali di valutazione. Di seguito il flusso logico e le scelte alla base di ciascun termine del modello:

1. **Baseline (BASE_MEDIA = 6.5)**
   Viene definito un "punto di partenza" comune a tutte le valutazioni, pari alla media scolastica italiana di riferimento.

2. **Difficoltà intrinseca della materia (MATERIA_DIFFICOLTA)**
   Ad ogni materia è associato un valore di difficoltà (es. –1.0 per Matematica, +0.7 per Scienze Motorie). Questo termine simula l'effetto che materie più ostiche tendono a produrre voti mediamente inferiori.

3. **Offset classe/docente**
   Per rappresentare le differenze di "clima di classe" e stili di insegnamento, a ogni coppia (id_classe, materia) viene assegnato casualmente un offset estratto da una gaussiana N(0, 0.4). Ciò permette a due classi con stesso indirizzo di avere votazioni leggermente diverse.

4. **Abilità generale dello studente**
   Ogni studente riceve un valore di "abilità" estratto da N(0, 0.6), che rappresenta competenze soft ed eclettiche non legate a una materia specifica.

5. **Specificità studente-materia**
   Per modellare inclinazioni individuali verso certe discipline, per ciascuna coppia (studente, materia) si genera un "bonus" da N(0, 0.3). Questo termine cattura, ad esempio, il talento personale in Latino o Fisica indipendentemente dall'abilità generale.

6. **Tipologia di valutazione**

   Il voto viene ulteriormente aggiustato in base al tipo di prova (scritto, orale, pratico). La funzione `tipologia_delta(materia, tipologia)` restituisce un delta variabile in base alla categoria della materia (HARD, UMANISTICA, LAB). Ad esempio, prove orali in materie umanistiche tendono a valorizzare di più l'espressività dello studente.

7. **Fattore socio-demografico**

   Si calcola un impatto unificato come media pesata (25% ciascuno) di quattro componenti:

   - Area geografica (es. Nord-Est +0.6, Sud –0.4),
   - Tipo di scuola (liceo +0.3, professionale –0.6),
   - Cittadinanza (UE –0.2, non-UE –0.6),
   - Quartile ESCS (dal –0.8 al +0.8).

   Questo termine riflette le differenze socio-economiche e culturali che influenzano il rendimento. [2, 3, 4, 5]

8. **Rumore casuale**

   Per evitare un'eccessiva omogeneità nei voti, viene aggiunto un "rumore" estratto da N(0, 0.7), garantendo naturale variabilità anche tra studenti simili.

Il voto grezzo è quindi:

```
v_raw = BASE_MEDIA
      + diff_materia
      + offset_classe
      + abilita_studente
      + spec_studente_materia
      + tipologia_delta
      + impatto_socio_demografico
      + noise
```

A questo punto, per evitare estremi troppo bassi o alti, il valore viene:

- Soft-floor a 3: se v_raw < 3 e con probabilità 60% viene "spostato" in [3,4], per non generare troppi insufficienze gravi;
- Clamping all'intervallo [1,10];
- Arrotondamento all'intero più vicino.

Infine, la data di registrazione del voto è scelta casualmente tramite la funzione `data_random()`, che estrae un offset di giorni uniformemente distribuito tra il 15/09/2023 e il 31/05/2024, assegnando a ciascun voto un timestamp realistico.

Questa combinazione di termini garantisce un'elevata fedeltà statistica: le medie aggregate risultanti per materia, quartile ESCS e area geografica replicano i pattern osservati nei report ufficiali, pur mantenendo la flessibilità e la riproducibilità necessari per esperimenti e test su database NoSQL.

5.4.7 Esportazione e verifica

Al termine, i record dei voti – composti da `id_voto`, `id_studente`, `id_docente`, `materia`, `voto`, `tipologia`, `data` – sono scritti in `voti.csv`.
Subito dopo, `analisi_dataset.py` ricompone studenti e voti per calcolare medie per cittadinanza, quartile ESCS e area geografica; la stampa console di queste tabelle conferma la coerenza del modello rispetto ai gap osservati nei report ufficiali `genera_dati_simulati`.

Con questa ricca combinazione di script, pesi empirici e generatori casuali controllati, NoSchoolQL_MIM realizza un dataset scolastico sintetico di elevata fedeltà, ideale per testare architetture NoSQL e per condurre analisi didattiche, statistiche e sociologiche.

CAPITOLO 6

Architettura del sistema

Il progetto NoSchoolQL_MIM è strutturato secondo un'architettura a tre livelli distinti, ciascuno con responsabilità chiare e interfacce ben definite. Questa scelta garantisce un'elevata coesione all'interno di ogni componente e un basso accoppiamento tra di essi, rendendo il sistema facilmente manutenibile ed estendibile nel tempo.

6.1 Vista high-level e flussi informativi

Al livello più alto, l'utente interagisce con una Single-Page Application React, che non detiene alcuno stato server-side: ogni interazione genera una chiamata RESTful al backend. Le richieste viaggiano su HTTP(S) in formato JSON, passando attraverso il CORS layer e il body parser di Express, quindi raggiungono i controller che implementano la logica di business. Dal controller, le operazioni di lettura e scrittura vengono inoltrate a MongoDB tramite il driver ufficiale, che a sua volta instrada le query verso il nodo primario del replica set. La replica asincrona verso i nodi secondari assicura tolleranza alle partizioni di rete e continuità di servizio.

Grazie a un design stateless sul server, ogni richiesta contiene tutte le informazioni necessarie (token JWT, parametri URL, body), permettendo scalabilità orizzontale spingendo più istanze di Express dietro a un bilanciatore di carico.

6.2 Backend (Node.js + Express)

Il backend si articola in:

- **Middleware chain**: gestisce in ordine CORS, parsing JSON, autenticazione e validazione. L'uso di token JWT semplifica la gestione delle sessioni, scaricando sul client la responsabilità del mantenimento dello stato.
- **Router layer**: smista le richieste sui diversi controller basandosi su prefissi modulari (`/api/statistiche`, `/api/registro`, `/api/home`), facilitando l'aggiunta di nuove aree funzionali senza impattare il codice esistente.
- **Controller layer**: ogni controller incapsula un caso d'uso, orchestrando la logica necessaria e delegando alla data access layer l'esecuzione delle query. Questa separazione permette di testare la logica di business in isolamento, con mocking del livello dati.
- **Data access layer**: interagisce con MongoDB tramite query documentali mirate, utilizzando indici su campi chiave (`id_studente`, `id_classe`, `id_docente`) per garantire rapidità nelle operazioni di ricerca e aggregazione.

L'architettura prevede inoltre un error handler centrale che intercetta eccezioni e restituisce al client un oggetto di risposta omogeneo, contenente uno status code HTTP appropriato e un messaggio descrittivo, semplificando la gestione dei messaggi di feedback nel frontend.

6.3 Frontend (React)

Il client React è stato progettato con un'architettura a componenti:

- **Page components**: container di alto livello che definiscono layout e data-fetching (es. Registro, Statistiche).
- **Presentational components**: elementi UI generici (bottoni, card, spinner, filtri) riutilizzabili in più contesti, ciascuno responsabile solo del rendering e delle callback verso il parent.

- **Context Provider**: AppContext gestisce lo stato globale di autenticazione, incluse le informazioni dell'utente e il token JWT, persistito in localStorage per sopravvivere ai refresh di pagina.
- **Async data layer**: un modulo centralizzato replica le chiamate REST, aggiunge automaticamente l'header di autenticazione e traduce gli errori HTTP in eccezioni gestibili dai componenti.
- **User experience**: durante il caricamento dei dati, vengono mostrati spinner o skeleton placeholder, mentre errori di rete o di validazione sono presentati con messaggi toast contestuali.

Il routing client-side definisce chiaramente quali percorsi sono pubblici (home, login, statistiche) e quali protetti (registro), utilizzando un wrapper ProtectedRoute che reindirizza al login in assenza di credenziali valide.

6.4 Database NoSQL (MongoDB)

MongoDB è stato scelto per la sua flessibilità e per il modello documentale adatto a gestire dati complessi e variabili, come quelli prodotti dalla pipeline Python. Le collezioni principali corrispondono ai dataset:

- **classi**: documenti con elementi anagrafici e aggregazioni demografiche;
- **studenti**: profili individuali con riferimento alla classe, ESCS e metadata;
- **docenti e assegnazioni_docenti**: separazione tra anagrafica docente e relazioni classe-materia;
- **voti**: record con chiavi esterne a studenti e docenti, tipi di valutazione e timestamp.

Pur non essendo imposto uno schema rigido, il progetto segue convenzioni di naming e strutture coerenti, favorendo embedding dove opportuno (es. campi demografici dentro classi) e referencing dove conviene (es. relazioni voto→studente). La replica set assicura availability e partition tolerance, mentre la segmentazione delle collezioni ottimizza le prestazioni di scrittura e lettura.

6.5 Deployment e ambiente operativo

Per l'esecuzione in ambienti diversi, il sistema fa uso di:

- **Variabili d'ambiente**: tutte le configurazioni sensibili (URI database, nome del database, segreto JWT, porta server) sono estratte da un file `.env` non versionato, isolando il codice dai dettagli di infrastruttura.
- **Script di avvio**: configurati in `package.json` per avviare il server in modalità produzione o sviluppo con reload automatico in caso di modifiche.
- **Continuous Integration**: un workflow automatizzato esegue la pipeline ETL Python, verifica lo stile del codice JavaScript con ESLint e compila il frontend, bloccando i merge in caso di errori.
- **Log centralizzati**: il server invia messaggi di log in console (con possibilità di redirect verso file o servizi esterni), tracciando eventi di connessione al database, avvio del server, operazioni CRUD e errori runtime.

Questo assetto, minimalista ma completo, assicura che il sistema possa essere distribuito, monitorato e aggiornato senza dipendenze esterne complesse, mantenendo elevati livelli di disponibilità e sicurezza.

CAPITOLO 7

Use Case e Requisiti Utente

Il capitolo dedicato agli use case e ai requisiti utente descrive i principali scenari d'uso dell'applicazione NoSchoolQL_MIM, con l'obiettivo di chiarire le funzionalità offerte sia agli studenti sia ai docenti, nonché i vincoli e le esigenze non funzionali che guidano il design del sistema.

7.1 Attori principali

- **Studente**
  - Accesso mediante ID (es. STU000001)
  - Consultazione dei propri voti e statistiche aggregate
- **Docente**
  - Accesso mediante ID (es. DOC00001)
  - Gestione completa delle valutazioni (inserimento, modifica, cancellazione)
  - Visualizzazione dell'elenco delle proprie classi e dei relativi studenti
- **Amministratore / Sistema**
  - Esecuzione automatica della pipeline ETL per rigenerare i dataset

– Monitoraggio del corretto funzionamento dei processi

7.2 Use Case Studente

7.2.1 UC1 – Login Studente

Lo studente accede inserendo il proprio ID nella pagina di login. Il frontend effettua una chiamata a

`GET /api/registro/studente/voti`

con header `Authorization: STUDENTE:<ID>`. Se la risposta è 200 OK, l'utente viene reindirizzato al registro elettronico in modalità studente.

7.2.2 UC2 – Visualizzazione Dashboard Studente

Una volta autenticato, lo studente visualizza una panoramica ("dashboard generale") che mostra:

- Grafico a barre delle medie per materia
- Distribuzione dei voti (istogramma)
- Media generale

Tutte le informazioni vengono caricate in parallelo via `Promise.all` da endpoint quali

```
GET /api/registro/studente/media-per-materia
GET /api/registro/studente/distribuzione-voti
GET /api/registro/studente/media-generale
```

e rese interattive con Chart.js.

7.2.3 UC3 – Dettaglio Voti per Materia

Selezionando una materia dalla sidebar, lo studente invoca

`GET /api/registro/studente/voti-materia/{materia}`

e ottiene la lista dei propri voti per quell'insegnamento, presentata con componenti `StudentVotoCard`.

7.3 Use Case Docente

7.3.1 UC4 – Login Docente

Il docente inserisce il proprio ID nel login form. Il client prova a recuperare le classi con

```
GET /api/registro/docente/classi
```

e, in caso di esito positivo, memorizza `tipo=docente` in localStorage e reindirizza alla dashboard docente.

7.3.2 UC5 – Selezione Classe e Materia

Dalla sidebar docente, l'insegnante seleziona una classe; il frontend filtra `classiDocente` per nome della classe, aggiorna `studentiClasse` richiamando

```
GET /api/registro/docente/classi
```

e individua solo gli studenti afferenti alla classe selezionata Registro.

Successivamente, seleziona la materia d'insegnamento, ottenendo l'elenco dei voti esistenti via

```
GET /api/registro/docente/voti?classe=<id_classe>&materia=<materia>
```

(implementazione nell'interfaccia `VotiList`).

7.3.3 UC6 – Inserimento Voti di Classe

Il docente, tramite `VotoClasseForm`, compone in un'unica operazione i voti di tutti gli studenti di una classe. Alla conferma, il client invia un payload JSON a

```
POST /api/registro/docente/classe/voti
```

contenente `id_classe`, `materia`, `data`, `tipo` e lista di `{ id_studente, voto }`. In caso di successo, la lista di voti viene ricaricata per riflettere le nuove valutazioni `genera_dati_simulati`.

7.3.4 UC7 – Modifica e Cancellazione di un Voto

Ogni voto è modificabile inline attraverso `VotiList`: l'utente può cambiare valore, tipologia o data. All'invio, il client effettua

`PUT /api/registro/docente/voto`

con body `{ id_voto, voto, tipologia, data }` per aggiornare il record, oppure

`DELETE /api/registro/docente/voto`

con `{ id_voto }` per eliminarlo. Il sistema risponde con status 200 OK e il frontend aggiorna dinamicamente l'elenco `VotiList`.

7.4 Requisiti Funzionali

1. **RF1 – Autenticazione**: login via ID, gestione token JWT;
2. **RF2 – Visualizzazione dati**: dashboard interattiva per studenti e docenti;
3. **RF3 – Gestione voti**: inserimento batch e modifica/cancellazione puntuale;
4. **RF4 – Statistiche**: grafici e tabelle per medie e distribuzioni;
5. **RF5 – Responsività**: UI fruibile su desktop e tablet.

7.5 Requisiti Non Funzionali

1. **RNF1 – Performance**: tempi di risposta API < 2 sec in condizioni normali;
2. **RNF2 – Scalabilità**: architettura stateless, possibile orizzontalizzazione di Express;
3. **RNF3 – Sicurezza**: trasmissione HTTPS, validazione JWT, CORS limitato all'origine frontend;
4. **RNF4 – Usabilità**: caricamenti asincroni con spinner/skeleton, messaggi di errore contestualizzati;
5. **RNF5 – Manutenibilità**: codice modulare, separazione netta tra logica di business, data access e presentazione.

CAPITOLO 8

Soluzione proposta e tecnologie

Nel progettare NoSchoolQL_MIM, l'obiettivo è stato coniugare flessibilità, scalabilità e rapidità di sviluppo, scegliendo un ecosistema tecnologico interamente JavaScript/Python per sfruttare competenze comuni e garantire una curva di apprendimento contenuta. Di seguito si illustrano le scelte architetturali, le librerie fondamentali e l'integrazione di componenti chiave come Faker e i sistemi di charting.

8.1 Scelte architetturali principali

La soluzione si articola in due macro-componenti:

1. **Pipeline ETL in Python**

   - I dati originali MIM vengono preprocessati e simulati con script indipendenti (`pulizia_mim.py`, `calcolo_statistiche.py`, `genera_dati_simulati.py`, `analisi_dataset.py`), orchestrati da `main.py`.
   - Questo separatore di responsabilità permette di isolare la logica di generazione/dati dal resto dell'applicazione e di riutilizzare facilmente la pipeline per aggiornamenti futuri o dataset differenti.

2. **Applicazione full-stack JavaScript**

   - Backend: Node.js con Express espone API REST, gestisce autenticazione via JWT e dialoga con MongoDB tramite il driver ufficiale.
   - Frontend: SPA React sviluppata con Vite, sfrutta React Router DOM per il routing client-side e un Context Provider per la gestione globale dello stato (autenticazione, dati utente).
   - Il tutto è stateless sul server, consentendo un rapido scaling orizzontale senza dipendere da sessioni in memoria.

8.2 Tecnologie e librerie utilizzate

- **Node.js & Express**
  - Runtime JavaScript leggero e non-bloccante, ideale per API ad alto I/O.
  - Express garantisce un framework minimale e modulare per definire middleware, route e controller.
- **MongoDB**
  - Database documentale NoSQL scelto per la sua natura schemaless, perfetto per ospitare i dataset generati (classi, studenti, docenti, voti) senza vincoli di schema rigido.
  - Configurato in replica set per alta disponibilità e tolleranza alle partizioni.
- **Python & Pandas**
  - Utilizzati nei processi ETL per la pulizia, normalizzazione e generazione dati; Pandas facilita operazioni tabellari e aggregazioni sulle colonne CSV.
- **Faker**
  - Libreria Python per generare nomi realistici (italiani e stranieri), cognomi e date casuali, arricchendo il dataset con dati di "dominio" plausibili.
- **React & React Router DOM**

  - React 18 con Vite per un bundler veloce e sviluppo hot-reload.
  - React Router DOM per gestire route pubbliche e protette in modo dichiarativo.

- **JWT**
  - JSON Web Token per l'autenticazione stateless; il client conserva il token in localStorage e lo invia con l'header Authorization ad ogni richiesta.

- **UI Components**
  - Architettura a componenti per bottoni, card, spinner, accordion e filtraggio, costruita internamente ma facilmente sostituibile con librerie come Material-UI o Chakra UI.

- **Charting**
  - Per la visualizzazione dei dati statistici (medie, distribuzioni), sono impiegate librerie JavaScript di charting (es. Chart.js), integrate in componenti React che consumano gli endpoint `/api/statistiche`.

8.3 Integrazione di Faker e sistema di charting

La scelta di Faker in Python permette di popolare i dataset con nomi e cognomi verosimili, garantendo varietà culturale (italiana, UE, non-UE) senza compromettere la privacy. Faker è inizializzato con un seed fisso (42), assicurando riproducibilità e coerenza tra diverse esecuzioni.

Sul fronte charting, il frontend recupera dati aggregati da endpoint dedicati (es. `/api/statistiche/area-geografica`, `/api/statistiche/escs-quartile`) e li passa a componenti che rendono istogrammi, grafici a barre e a torta. Questi componenti:

- Supportano aggiornamenti dinamici e reattivi a filtri di periodo o materia.
- Gestiscono layout responsive e callback di selezione per mostrare dettagli contestuali.

8.4 Aggiornamenti on-demand dei dati

La sezione dedicata agli aggiornamenti real-time non si applica: l'applicazione non utilizza né polling né WebSocket, ma si affida completamente a richieste on-demand per mantenere la UI sincronizzata con il backend.

In pratica, ogni volta che l'utente accede a una vista — ad esempio la pagina delle statistiche o il registro elettronico — il componente React esegue al montaggio una serie di chiamate HTTP per recuperare i dati correnti. All'interno di un hook `useEffect`, con dipendenze sul contesto di navigazione (classe selezionata, materia, filtri, etc.), il client invoca gli endpoint necessari e aggiorna lo stato locale non appena la risposta giunge: questo assicura che non vengano mai visualizzate informazioni obsolete.

Analogamente, quando il docente inserisce, modifica o elimina un voto, l'operazione POST/PUT/DELETE viene seguita immediatamente da una nuova fetch sull'elenco aggiornato dei voti di quella classe e materia. In questo modo l'interfaccia riflette in tempo reale le modifiche, senza dover ricorrere a meccanismi di sincronizzazione automatica: il flusso è sempre innescato da un'azione utente o dal ciclo di vita del componente.

Questa strategia semplifica l'architettura, mantenendo il server completamente stateless e riducendo il carico di richieste superflue, pur garantendo un'esperienza reattiva e coerente con le aspettative dell'utente.

CAPITOLO 9

Metodologia di sviluppo

La metodologia adottata in NoSchoolQL_MIM segue un approccio iterativo, modulare e basato su strumenti di automazione leggeri, garantendo che ogni fase – dall'analisi dei requisiti alla manutenzione del codice – sia chiaramente definita e ripetibile.

9.1 Analisi e raccolta requisiti

Il punto di partenza è stato un'esplorazione sistematica dei dataset MIM originali, senza alcuna inferenza esterna: si sono caricati i CSV con Pandas per esaminare struttura e contenuti delle tabelle di anagrafica scuole, studenti per indirizzo e cittadinanza, nonché ordinamento per corso di studi. Da questi file sono stati ricavati i requisiti funzionali (es. operazioni CRUD, filtri per materia/periodo) e non funzionali (performance, scalabilità, sicurezza JWT) che hanno guidato il design dell'intera soluzione.

Parallelamente, la scelta dei parametri socio-demografici (range ESCS [3], impatti geografici, differenze tra tipi di scuola e cittadinanza, quartili) è stata direttamente codificata negli script di simulazione, attingendo a valori ufficiali come:

- `ESCS_MIN`, `ESCS_MAX` definiti nel Decreto MI 220/2022 [4]
- `GEOGRAFIA_IMPACT` e `CITTADINANZA_IMPACT` basati sui report INVALSI 2024 [2]

- `TIPO_SCUOLA_IMPACT` e `MATERIA_DIFFICOLTA` da esiti Maturità 2023/2024 [5]

9.2 Design tecnico

Il progetto è stato scomposto in quattro script Python indipendenti – `pulizia_mim.py`, `calcolo_statistiche.py`, `genera_dati_simulati.py` e `analisi_dataset.py` – ciascuno con una responsabilità univoca e orchestrati da `main.py` che esegue sequenzialmente ogni fase. Questo design a pipeline:

1. Garantisce isolamento delle logiche di trasformazione dati,
2. Permette test incrementali, fermandosi alla prima fase che incontra un errore,
3. Favorisce la manutenibilità, consentendo di aggiornare o sostituire uno script senza impattare gli altri.

Sul versante JavaScript, il backend Express è stato organizzato in middleware, router e controller, mentre il frontend React adotta un pattern "Page vs Presentational Components" con un contesto globale per l'autenticazione (`AppContext.jsx`) e un router protetto (`ProtectedRoute.jsx`).

9.3 Implementazione

9.3.1 Pipeline ETL

- **Pulizia**: `pulizia_mim.py` carica i CSV originali, normalizza stringhe, applica filtri per isolare solo le scuole di secondo grado e produce file puliti in `file/dataset_puliti`.
- **Statistiche di base**: `calcolo_statistiche.py` legge i dati puliti e calcola percentuali di genere e cittadinanza, generando `statistiche_base.csv`.
- **Generazione simulata**: `genera_dati_simulati.py` utilizza i dataset intermedi e i parametri socio-demografici per creare le collezioni documentali di classi, studenti, docenti e voti, salvate in `file/dataset_definitivi`.

- **Analisi finale**: `analisi_dataset.py` confronta i voti simulati con le statistiche di base per validare la coerenza dei modelli.

Ogni script è strutturato con funzioni pure (es. `genera_escs_studente`, `calcola_socio_demografico`, `voto_generato`) e un blocco `if __name__ == '__main__'` per permetterne l'importazione e il test isolato.

9.3.2 Applicazione full-stack

- **Server**: `server.js` avvia Express solo dopo la connessione a MongoDB (`db/connection.js`), monta middleware globali (CORS, JSON parsing) e registra router modulari per statistiche, registro e homepage.
- **Client**: React con Vite, utilizza `useEffect` per il fetch on-demand dei dati e `useState/useContext` per gestire sessione e risultati; i componenti UI (card, filtri, spinner) sono isolati in file separati per favorirne il riuso.

9.4 Verifica e testing

In assenza di una suite automatica, si è adottato uno smoke testing manuale ma rigoroso:

1. **ETL completo**: esecuzione di `python main.py` con successiva verifica della presenza di tutti i CSV e controllo delle intestazioni e dei conteggi risultanti.
2. **API testing**: utilizzo di un REST Client (es. estensione VSCode) o di comandi curl per collaudare endpoint CRUD, verificando status code (200, 201, 400, 401) e formato JSON restituito.
3. **UI end-to-end**: avvio simultaneo di backend e frontend in locale, passo a passo attraverso i casi d'uso (login, fetch dati, inserimento/modifica voti, filtri), osservando la coerenza del flusso e l'accuratezza dei risultati.

Le uscite di `analisi_dataset.py` (medie per cittadinanza, quartile ESCS, area geografica) sono state confrontate con i gap noti dalla letteratura, confermando l'allineamento del modello generativo.

9.5 Refactoring e mantenimento

Pur essendo gestito da un singolo sviluppatore, il progetto beneficia di refactoring continuo:

- Suddivisione di funzioni troppo lunghe in helper autonomi (ad es. separazione di `voto_generato` in calcolo di abilità, offset, socio-demografico).
- Documentazione inline con docstring e JSDoc per chiarire parametri, return value e side-effect.

Grazie a questa metodologia, NoSchoolQL_MIM garantisce qualità del codice, facilità di debugging e rapidità di evoluzione del progetto, anche in un contesto di sviluppo monopersonale.

CAPITOLO 10

Best Practice e Sicurezza

La qualità e la robustezza dell'applicazione NoSchoolQL_MIM dipendono dall'adozione di pratiche consolidate e da accurate misure di sicurezza, applicate sia alla pipeline ETL in Python sia alla componente full-stack JavaScript. Di seguito le raccomandazioni effettivamente seguite, senza riferimenti a tool non configurati nel progetto.

10.1 Principi di codifica e manutenzione

- **Separazione di responsabilità**
  Ogni script Python e ogni modulo JavaScript ha un ambito ristretto e chiaramente definito. Ad esempio, in `genera_dati_simulati.py` la logica di generazione dei voti è suddivisa in funzioni come `calcola_socio_demografico`, `tipologia_delta` e `voto_generato`, evitando funzioni monolitiche.

- **Funzioni puramente funzionali**
  Le operazioni critiche (calcolo ESCS, voto simulato, conversioni sicure) sono implementate come funzioni che dipendono esclusivamente dai parametri di input, restituendo sempre output deterministici per semplificare refactoring e debug.

- **Code review personale**

  Pur essendo un progetto monopersonale, ogni modifica rilevante viene esaminata "a caldo" rivedendo i diff prima del merge su main, assicurando coerenza stilistica e correttezza logica.

10.2 Gestione di configurazioni e credenziali

- **File .env**

  Tutte le impostazioni sensibili (URI MongoDB, nome del database, segreto JWT, porta server) risiedono in un file `.env` che non è mai stato committato nel repository, caricato in `server.js` tramite dotenv.

- **Zero hard-coded secrets**

  Non sono presenti nel codice password, token o chiavi: ogni segreto è ottenuto da `process.env` (backend) o da variabili d'ambiente (ETL Python), evitando il rischio di leak accidentali.

10.3 Sicurezza delle API e convalida degli input

- **Autenticazione e autorizzazione**

  Il middleware `authMiddleware.js` verifica la validità del token JWT estratto dall'header Authorization e arricchisce `req.user`. In base al ruolo (studente o docente), i controller limitano o consentono operazioni di modifica sui dati.

- **Validazione dei payload**

  Ogni endpoint che crea o modifica dati controlla la presenza e il formato dei campi essenziali (es. `id_studente`, `voto`, `data`). In Python, la funzione `to_int_safe` gestisce conversioni di tipo senza generare eccezioni.

- **Filtro CORS**

  In `server.js`, la configurazione CORS è pensata per essere limitata, consentendo richieste soltanto dalle origini del frontend in produzione.

10.4 Protezione dei dati e conformità

- **Trasporto cifrato**
  Pur non gestendo direttamente i certificati, l'applicazione è progettata per girare dietro HTTPS, garantendo la cifratura dei dati in transito.

- **Minimizzazione delle informazioni restituite**
  Gli endpoint restituiscono soltanto i campi necessari al client, escludendo qualsiasi dato superfluo o sensibile non richiesto.

10.5 Logging e auditing

- **Log strutturati**
  Il backend utilizza `console.log` per eventi significativi (connessione DB, avvio server, errori runtime). Questi log, stampati in console, possono essere reindirizzati a file o sistemi di raccolta esterni.

- **Tracciatura delle modifiche**
  Ogni record di voto include `id_voto` univoco e data, permettendo di ricostruire l'ordine cronologico delle operazioni senza necessità di un sistema di versioning dedicato.

10.6 Hardening dell'ambiente di esecuzione

- **Esecuzione con utente ridotto**
  Si raccomanda di avviare i processi Node.js e Python sotto un utente non privilegiato, limitando i rischi in caso di compromissione.

- **Backup regolari**
  I dati di MongoDB e i CSV generati dalla pipeline ETL devono essere sottoposti a backup periodici, garantendo ripristini rapidi in caso di guasti o cancellazioni accidentali.

Attraverso queste pratiche, NoSchoolQL_MIM assicura un equilibrio tra semplicità operativa e solidità, proteggendo i dati degli utenti e mantenendo un codice chiaro e affidabile.

CAPITOLO 11

Conclusioni

Al termine della realizzazione di NoSchoolQL_MIM, si evidenziano i principali risultati conseguiti e le possibili linee di evoluzione, con l'obiettivo di arricchire le funzionalità, migliorare le prestazioni e ampliare il dominio di applicazione.

11.1 Sintesi dei risultati

1. **Pipeline ETL robusta**

    - Grazie alla suddivisione in script indipendenti, è stato possibile pulire i dati grezzi MIM, calcolare statistiche di base, simulare studenti, classi, docenti e voti con fattori socio-demografici realistici, e validare il modello attraverso analisi finali.
    - L'uso di un seed unificato (SEED = 42) assicura riproducibilità totale degli output.

2. **Architettura full-stack modulare**

    - Il backend Express, organizzato in middleware, router e controller, è completamente stateless e pronto per l'orizzontalizzazione.

- Il frontend React, basato su Vite, offre un'esperienza utente fluida con componenti riutilizzabili e routing protetto.

3. **Modello dati NoSQL efficace**

   - Le collezioni documentali (classi, studenti, docenti, voti) sono state progettate per supportare query frequenti su ID studente, classe e docente, garantendo buone performance senza schema fisso.

4. **Allineamento con evidenze ufficiali**

   - I pesi socio-demografici e le difficoltà delle materie sono stati impostati in base a parametri tratti direttamente dai report INVALSI[2], dal Decreto MI 220/2022[4] e dagli esiti di Maturità 2023/2024[5], senza mediazioni intermedie.

11.2 Limiti attuali

- **Assenza di test automatici**
  La mancanza di unit/integration test limita la garanzia di regressione automatica; oggi il progetto si basa su smoke test manuali.

- **Capacità di scaling del pipeline ETL**
  L'esecuzione dei quattro script in sequenza è efficiente per dataset di alcune centinaia di scuole, ma potrebbe diventare onerosa con volumi più elevati.

- **Funzionalità di audit limitata**
  Sebbene ogni voto riporti data e ID univoco, manca un vero sistema di versioning o log di tutte le operazioni per un audit completo.

11.3 Conclusione

Il framework realizzato rappresenta un proof-of-concept solido per la generazione e la gestione di dataset scolastici sintetici in ambiente NoSQL, integrando evidenze ufficiali e garantendo riproducibilità. Le proposte di miglioramento, se implementate,

consentiranno di trasformarlo in una piattaforma ancora più completa, adatta a scenari di ricerca accademica, analisi dei dati educativi e sperimentazioni su larga scala.

Bibliografia

[1] ISO/IEC/IEEE, *Systems and software engineering — Requirements for managers of information for users of systems, software, and services*, International Organization for Standardization International Standard ISO/IEC/IEEE 26 511:2018, 2018, published jointly by ISO, IEC and IEEE. (Citato a pagina 2)

[2] INVALSI, "Rapporto nazionale invalsi 2024," INVALSI - Istituto nazionale per la valutazione del sistema educativo di istruzione e di formazione, Roma, Italia, Tech. Rep., 2024. [Online]. Available: https://www.invalsiopen.it/risultati/risultati-prove-invalsi-2024/ (Citato alle pagine 17, 19, 33 e 42)

[3] ——. (2021) L'indicatore escs per una valutazione più equa. Articolo divulgativo sul sito INVALSI Open. [Online]. Available: https://www.invalsiopen.it/indicatore-escs-valutazione-equa/ (Citato alle pagine 17, 19 e 33)

[4] Ministero dell'Istruzione, "Decreto n. 220 dell'8 agosto 2022 – adozione degli indicatori escs," Aug. 2022, gazzetta Ufficiale e MIUR, Registro Decreti 0000220.08-08-2022. [Online]. Available: https://www.miur.gov.it/documents/20182/0/Decreto+220+indicatori+ESCS.pdf (Citato alle pagine 17, 19, 33 e 42)

[5] Ministero dell'Istruzione e del Merito, "Esiti degli scrutini e degli esami di stato 2023/2024," Ministero dell'Istruzione e del Merito, Roma, Italia, Tech. Rep., 2024, dati rilevati al 29 luglio 2024, fonte: Anagrafe Nazionale Studenti. (Citato alle pagine 17, 19, 34 e 42)

[6] ——, "Gli alunni con cittadinanza non italiana a.s. 2022-2023," Ministero dell'Istruzione e del Merito, Roma, Italia, Tech. Rep., 2024, focus pubblicato in collaborazione con l'INVALSI. [Online]. Available: https://www.miur.gov.it/-/gli-alunni-con-cittadinanza-non-italiana-a-s-2022-2023 (Citato a pagina 17)